This thesis is submitted to the Faculty of Computer Science & Information Technology, University of Malaya in partial fulfillment of the requirements for the Bachelor of Computer Science Degree.

Supervisor : En. Noorzaily Mohamed Noor

Moderator : En. Yamani Idna Idris

Abstract

As part of fulfillment of the requirements for the Bachelor of Computer Science Degree, we are required to do a thesis on a selected title. My thesis title is Smart Sensor Simulator.

This project will be on smart sensors that are based on IEEE 1451.2 standard. Smart sensors are replacing the traditional sensors that were once used. The IEEE 1451 standards specify sets of common communication interfaces to standardize the connectivity of transducers to microprocessor, instrumentation systems, and networks. This is done by specifying different standards that address the various aspects of the development of smart networked transducers. The scope of my thesis was narrowed down to doing a simulation of the electronic data sheet that is associated with the smart sensors.

The electronic data sheet keeps the transducers information. A simulation of how the data sheet keep and retrieve the transducers information to and from the RAM and Flash memory will be done. Hopefully this project will widen my knowledge on smart sensors and serves as a test bench for those who want to enhance the project.

Acknowledgement

I would like to extend my sincere appreciation and greatest gratitude for the following people:

a. My supervisor, Mr. Noorzaily for his advice and generous guidance whilst supervising this thesis.

b. My moderator, Mr. Yamani for not hesitating whenever I had question for him regarding my thesis.

c. My friends Savin, Ain, Ana, Marina and other course mates who have helped me and positively challenged me to create the best research and thesis project.

d. My family for giving me continuous motivation and support throughout this project.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

Ack	acknowledgment
ADC	analog-to-digital converter
ASIC	application-specific integrated circuit
DAC	digital-to-analog converter
DI/O	digital input/output
EEPROM	electrically-erasable programmable read-only memory
FPGA	field-programmable gate array
ID	identification
lsb	least significant bit
msb	most significant bit
NaN	not a number
NCAP	Network Capable Application Processor
r/w	read/write
SI	International System of Units
STIM	Smart Transducer Interface Module

TEDS	Transducer Electronic Data Sheet
TII	Transducer Independent Interface
UTC	Universal Coordinated Time
xdcr	transducer

Chapter 1 : Introduction

1.0 Chapter introduction

This chapter begins with the basic definition of the smart sensor simulator project suggested. The project that was going to be built is the simulator for a Smart Transducer Interface Module (STIM) for sensors and actuators. It also addresses the needs for a STIM in industries. The motivations for the project are also outlined. Various aspects of the project such as scope and limitations are also drawn up. Finally, a thorough project schedule was created for the entire development process of the project.

1.1 Project introduction

The increasing miniaturization of sensors and actuators (referred to as *transducers*) has led to an explosion in demand for small, cheap transducer networks. The term *smart* is used to refer to transducers with a certain level of performance beyond simple data input/output. There are a lot of advantages to smart, networked sensors including reducing wiring, self-identification, and auto-configuration. Self-identification simplifies the configuration and management of sensor networks. The deployment problem however remains communication between the central manager and the individual transducers. There are many different networking protocols in use today, from GPIB, PODBUS and RS232 to Ethernet and AppleTalk. Transducer may be connected to any one of these networks for a particular application. Because of this the manufacturer does not know which to support. Supporting multiple protocols is often a

necessity to remain competitive, but the demands on developers can be high as often a new protocol means a complete reimplementation of the transducer interface.

This is where IEEE 1451 enters the picture. IEEE 1451 is a family of proposed standards for *"A Smart Transducer Interface for Sensors and Actuators"*. Together these standards provide a single common interface between a transducer and external network, independent of the network protocol in use. They cover network capable smart transducers from the basic transducer interface up to a high-level, object-model representation of their behavior and communication. The goal of the specifications is to separate the design of the sensors and actuators from that of the networking controller, and to make the network protocol transparent to the transducer.

The five current sub-proposals are:

- **1451.2**: *"Transducer to Microprocessor Communication Protocols and Transducer Electronic Data Sheet"*. This specification defines the interface between the transducer hardware, called the STIM (smart transducer interface module), and the network controller, called the NCAP (network capable application processor). A sensor vendor has simply to implement support for the STIM side of the 1451.2 interface and then any type of external network support may be used through an appropriately chosen NCAP.
- **1451.1**: *"Network Capable Application Processor Information Model"*. This proposal is intended to develop the NCAP into higher level blocks (physical, network, functional, transducer) for implementation and to develop a standardized software interface for connecting the NCAP to a control network.

- **1451.3**: This draft proposal is an attempt to develop a multi-port version of 1451.2, allowing a single STIM and NCAP to control multiple transducers by sharing a single bus. It has not yet been approved by the IEEE.
- **1451.4**: This draft proposal is intended to define an interface specification for analog transducers. It is in the very early stages of development.
- **1451.5**: This draft proposal is intended to define a wireless transducer interface. It is also in the very early stages of development.

However the focus of this project is on IEEE 1451.2, STIM which will be defined later in the chapter. The simulator for a STIM will be built and tested so that it will a give a representation of what it would be like in real life.

1.2 Project definition

IEEE 1451.2 STIM stands for Smart Transducer Interface Module and it defines a digital interface and serial communication protocol which allows any transducer, or group of transducers, to receive and send digital data using a common interface. This common interface, called the "Transducer Independent Interface" (TII), is a 10-wire serial I/O bus that is similar in many ways to the IEEE 488 bus. The TII implements a serial data exchange with allowances for handshaking and interrupts, has defined power supply lines and permits hot-swapping of modules for plug-and-play capability. Any transducer can be adapted to the 1451.2 protocol with a Smart Transducer Interface Module, or STIM. This building block of the 1451 standard is the heart of the measuring system. It can be as simple as a switch connected to a 4-bit processor, or as complex as a 255 channel device running an individual process. The STIM performs functions like

signal conditioning, signal conversion and linearization. If added extra hardware it can perform functions such as spectrum analysis, fuzzy pattern recognition, adaptive noise canceling or any specific algorithm needed. The success of IEEE P1451 depends on the development of STIM's to meet individual needs and special applications.

A STIM is a module that contains TEDS, logic to implement the transducer interface, the transducer(s) and any signal conversion or signal conditioning. It consists of 1 to 255 sensors or actuators or any combination of them, DAC, ADC and Digital I/O to interface the sensors or actuators. It also consists of conversion circuitry, address logic, and digital electronics or microprocessors to convert the sensor readings into digital form, or to convert digital output to manipulate actuators to and from the network. The address logic facilitates communication between the STIM and an NCAP. Through the NCAP sensor data is passed to a network. When a STIM contains more than one transducer, it may be referred to as a multichannel STIM or a multi variable STIM.

1.3 Project motivation

Motivations of this project are:

- To have a deeper understanding of how STIM works in a real environment.
- To understand how STIM can simplify the connectivity of transducers to control system or networks.
- To learn how STIM can allow sensor manufacturers/ users to support multiple control network.

- To understand the ability of self-identification of the transducer to the system.
- To understand how the TEDS data are encoded and kept in Flash memory.

1.4 Project objectives

The objectives of this project are:

- To build the simulation of how the TEDS data are read and write into the RAM and FLASH.
- To understand how the TEDS data are encoded.

1.5 Expected outcome

A simulation of how the TEDS data are transmitted to and from Flash memory will be seen.

All the information of the virtual transducers can be stored retrieved from the TEDS flash area. Information such as manufacturer ID, calibration data, worst-case channel scenario, lower range limit and upper range limit can be viewed in hexadecimal form. As a result this interface appears to be unfriendly and suitable for diagnostic purposes only.

1.6 Project scope

The scope of this project to build the TEDS (Transducer Electronic Data Sheet) and a simulation of how the TEDS data will be kept and retrieved to and from Flash memory and RAM. The storage type for TEDS will be Flash memory type. The TEDS will consist of 2 channels which are for the actuator and sensor. There are 8 types of

TEDS data blocks namely Meta-TEDS, Channel TEDS, Calibration TEDS, Meta-Identification TEDS, Channel Identification TEDS, Calibration Identification TEDS, End-Users' Application-Specific TED, and Generic Extension TEDS. The Meta-TEDS and the Channel TEDS are mandatory while the rest are optional. In this project, only the Meta-TEDS and the Channel TEDS will be implemented. The sensor that will be used is a temperature sensor and the actuator will be a fan. Both the transducers will be virtual in the sense that the transducers are dummy not real.

1.7 Project limitations

Only unsigned data will be done to be entered in the Flash memory. There will no floating points or signed data. Characters are also not included. Basic function such as read, write and clear will be done for the Flash memory.

1.8 Project development schedule

This report covers the all the chapters of WXES 3181 and WXES 3182, which began on 23rd June 2004 and was completed on 25th February 2005. A total of ten key activities have been completed throughout the project comprising of the following (Table 1.0) :

No.	Key activities	Duration (weeks)
	WXES 3181	
1.	Introduction & objective study.	Week 1 – Week 2
2.	Literature review study.	Week 3 – Week 6
3.	Methodology planning.	Week 7 – Week 8
4.	System analysis	Week 8 – Week 10
5.	System Design	Week 10- Week 12

6.	Presentation (WXES 3181)	Week 12- Week 13
	WXES 3182	
7.	System Implementation	Week 1- Week 7
8.	System Testing	Week 8 – Week 10
9.	System Evaluation & Conclusion	Week 11-Week 12
10.	Presentation (WXES 3182)	Week 13- Week 14

Table 1.0 : Project development allocation.

The following Gantt chart (Figure 1.0 and Figure 1.1) shows the schedule of the whole project that is WXES 3181 and WXES 3182.

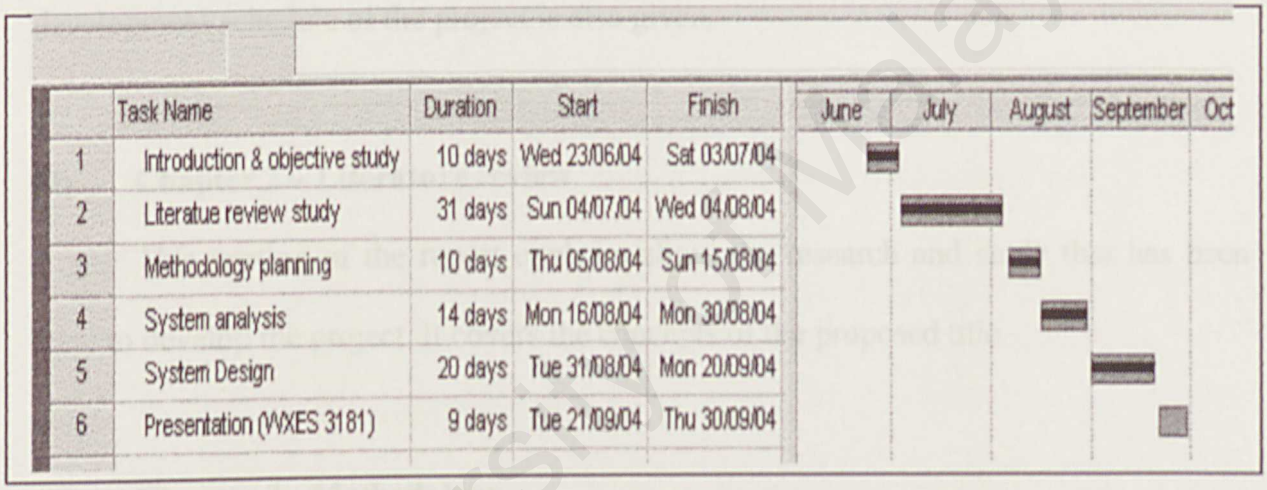

Figure 1.0 : Project development Gantt chart for WXES 3181

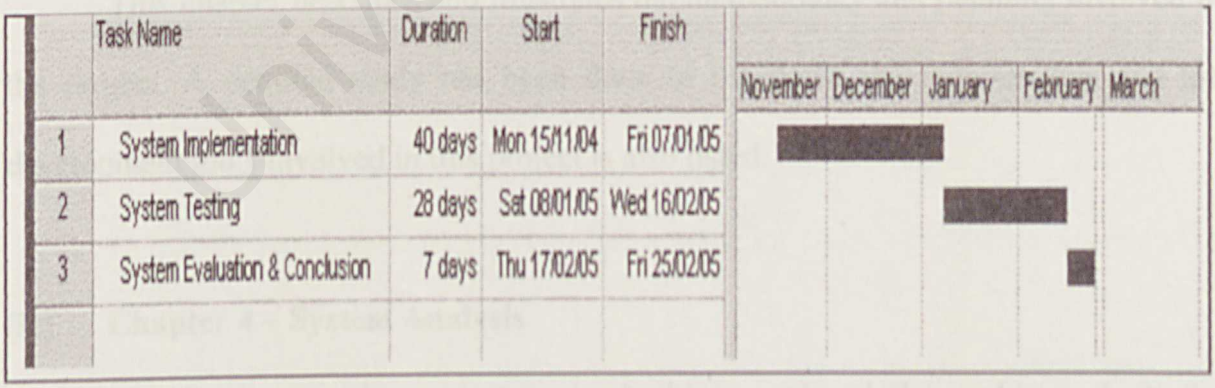

Figure 1.1 : Project development Gantt chart for WXES 3182

1.9 Report layout summary

This report covers the proposals, project development methodology scheme involved in the development of this project, system implementation, system evaluation and others as outlined below:

(i) Chapter 1 – Introduction

This section of the project entails an overview of the project. A conclusive of objectives of the project has been drawn up. In addition, the limitations and the development schedule of the project is also given.

(ii) Chapter 2 – Literature review

This portion of the report explains about the research and study that has been done to develop the project. It covers the concepts of the proposed title.

(iii) Chapter 3 - Methodology

This chapter describes and illustrates the methodology and planning involved in the project. A detailed study has been done to list down the requirements and the development tools involved in this project is also listed.

(iv) Chapter 4 – System Analysis

In this chapter, the project to be build is analyzed thoroughly and all the requirements needed in order to build the project is listed. It covers non-functional and also hardware requirements.

(v) Chapter 5 – System Design

This section reports the various aspects of the system design phases in this project. All the development modules are listed and appropriate steps are taken to design and build the modules.

(vi) Chapter 6 – System Implementation

This section discusses about how the system was implemented. It includes about the programming language used, some of the coding, and the coding style.

(vii) Chapter 7- System Testing

This chapter discusses about the testing that was done throughout the whole project.

(viii) Chapter 8 – System Evaluation & Conclusion

This chapter includes about the evaluation of the system which includes objectives achieved, system enhancement, problems faced, and discussion

1.10 Chapter summary

Hopefully this project will be a stepping stone for those who want to enhance the project.

Chapter 2 : Literature Review

2.0 Chapter introduction

This chapter entails the overall study of sensors and its counterparts. This chapter will begin with the importance of networking the transducers and the barriers for transducers manufacturer. The functional capability of a smart sensor will also be explained.

Going on, the chapter will discuss about the IEEE history and the available standards and why is there a need for them in the industry/government and why are they being developed.

Finally the existing STIM will be discussed and analyzed. The existing STIM will also be compared with the proposed project.

2.1 Smart transducers

A smart transducer has to provide more capabilities other than simply correct representation of a sensed or controlled physical quantity. According to IEEE 1451 drafts, a transducer is intelligent if it contains extra functionality that simplifies the integration of the transducer into any networked environment. Another smart feature is the ability of self-identification of the transducer to the system. This is done by providing the capability to embed key information about the transducer and its performance in a standardized format in a small amount of nonvolatile memory associated with the transducer. At power up or query from the system the transducer can identify to the host processor. Standardization of these features increases interoperability.

2.1.1 Transducer interface problem

There are many barriers for sensor manufacturers. This includes:

- Large number of different networks to support.
- Lack of network software know-how and support.
- Significant sensor interface software development for each supported network.
- Lack of a standardized sensor interface.

Because the transducer and the network must expose their two interfaces directly to the application, any trial to migrate the application to another platform is just cause for a complete redesign of the application's interface to the new environment. Transducer manufacturers and system integrators currently struggle with these issues. The transducer products take a longer time to market with higher price tags because the redesign process is time-consuming and expensive.

Transducer manufactures expand their engineering effort to cover several control network vendor technologies instead of designing a device once for all networks that supports the standardized interfaces.

So there came a need to network the sensors.

Networking smart sensors enable features not readily available with traditional sensors:

- The simplified wiring lowers the total system cost
- "Time aware" for time-stamping function
- Local networking to share measurement and control
- Provide Internet connectivity, thus you can access the "sensor information" from anywhere.

Functional capability of a smart sensor:

- integrated intelligence closer to the point of measurement and control
- basic computation capability
- capability to communicate data and information in a standardized digital format (self identification to the system)

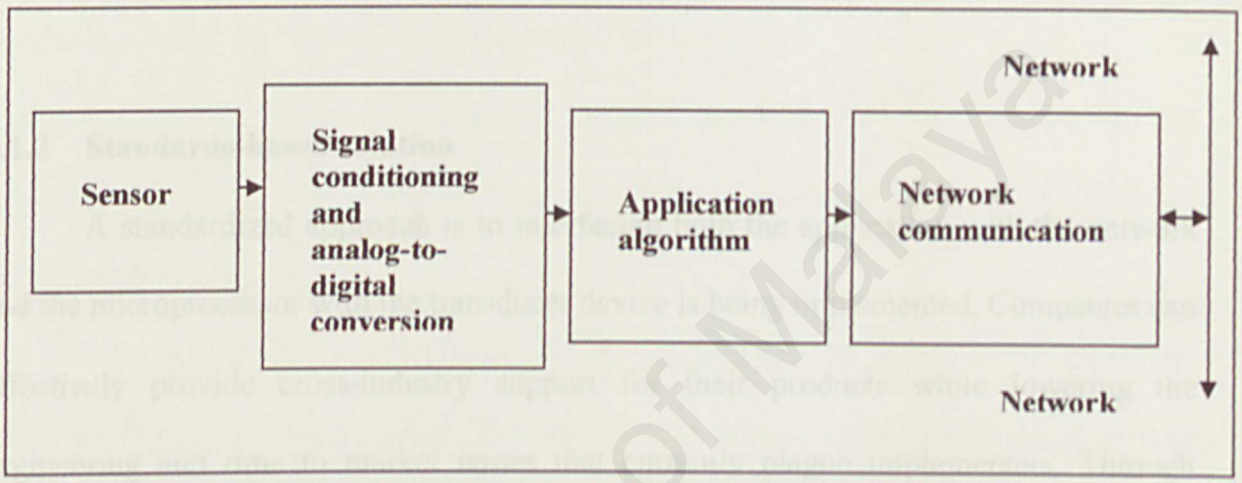

Figure 2.0 : Integrated Networked Smart Sensor

Example needs in government/industry.

Networked sensors are needed in:

- Example 1: U.S Navy needs tens of thousands of networked sensors per vessel to enhance automation because of the reduced-manning program.
- Example 2: Boeing needs to network hundreds of sensors to monitor and characterize airplane wiring performance.
- Example 3: Another example of a smart sensor network configuration is in computing, and other microprocessor-controlled applications, where temperature sensors need to communicate directly with the processor to prevent overheating

problems and failures. To address this application, Andigilog (Phoenix, AZ) has designed a monolithic semiconductor junction temperature sensor that includes a serial two-wire digital output. The temperature sensor's analog signal is converted to a digital format by an on-board delta-sigma converter. Communication is accomplished via a simple two-wire, SMBus-compatible serial I/O port, a ubiquitous serial interface for computer equipment.

2.1.2 Standards-based solution

A standardized approach is to interfacing both the application with the network and the microprocessor with the transducer device is being implemented. Companies can effectively provide cross-industry support for their products while lowering the engineering and time to market issues that currently plague implementers. Through standardized or common interface, the same transducers can be used on multiple control networks.

Smart features are being integrated into sensors and actuators (transducers). This uses of digital communication and networked configurations for connecting sensors and actuators is increasing. Distributed sensing and distributed control architecture is the trend now. The networked sensor technology that is applied to commercial and consumer applications includes aerospace and automotive, industrial automation, process control, smart buildings and home.

Smart sensors are replacing traditional sensors, exploiting the cost reduction trend of digital electronics. In fact, smart sensors are provided by an additional microcontroller unit and they could transform the raw sensor signal into a more

practical- to-use form. Signal conditioning, digital conversion, and processing might be included to allow an easier way to interface with higher level systems. However main innovation that smart sensors offer is the network capability. Most probably the future will show many small but clever sensors that share information using a generic network arrangement.

IEEE 1451 standards have recently ruled such standards. The aim of this standard, which is composed of four parts (1451.1, 1451.2, P1451.3, P1451.4), is to enable plug and play at transducer level, standardizing data structures and communication, and to simplify the creation of networked sensor model over a network independent system.

However IEEE specifies at least two layers between the transducer and the network: first, the 1451.2 Smart Transducer Interface Module (STIM) to interface sensors and hold their TEDS data (Transducer Electronic Data Sheet); second, the 1451.1 Network Capable Application Processor (NCAP) that supports network communications and it is able to run programs locally and to cooperate with other NCAPs.

2.2 History IEEE 1451

IEEE 1451 is a new family of standards (and proposed standards) for connecting smart transducers to networks. The first standard to be published was IEEE 1451.2, which was approved in September of 1997. So what is IEEE 1451.2?

- It is the specification of an electronic data sheet;

- It **is** the specification of a digital interface to access that data sheet, read sensors, and set actuators; and
- It **is not** another field network; it is an open standard which may be used with different networks.

IEEE 1451.1 describes the network-level, object-oriented model of 1451 devices. IEEE 1451.1 also has been approved and is going through the formal process of being published as an IEEE standard.

Its purposes are to:

- Provide a general transducer data, control, timing, configuration and calibration model.
- Provide a set of common interfaces for connecting sensors and actuators to instruments, and control and field networks.

Main goals for 1451:

- Develop network independent and vendor independent transducer interfaces.
- The transducers can be replaced/moved with minimum effort.
- Decrease error prone, manual system configuration steps.
- Support a general transducer data, control, timing, configuration and calibration model.
- Develop Transducer Electronic Data Sheets that remain together with the transducer during normal operation.

What standards are being developed?

- IEEE standard 1451.1-1999, Network Capable Application Processor (NCAP) Information Model for smart transducers—Published standard.
- IEEE 1451.2-1997, Transducer to Microprocessor Communication Protocols and Transducer Electronic Data Sheet (TEDS) Formats – Published standard.
- IEEE P1451.3, Digital Communication and Transducer Electronic Data Sheet (TEDS) Formats for Distributed Multidrop Systems – Being developed.
- IEEE P1451.4, Mixed–mode Communication Protocols and Transducer Electronic Data Sheet (TEDS) Formats – Being developed.

Benefits from 1451:

For Sensor manufactures

- Different products may be developed just by changing the TEDS.
- Standard physical interfaces.
- Standard calibration specification

For System integrators

- Self-documenting hardware and software.
- Systems that is easier to maintain.
- Rapid transducer replacement.
- Mechanism to store installation details.

For Application software programmers

- Standard transducer model for control and data.
- Same model for accessing a wide variety of measurements.
- "Hooks" for synchronization, expectations, simultaneous sampling.
- Support for multiple languages.

For End users

- "Plug and Play" sensors.
- Analysis software that can automatically provide:
  - physical units
  - readings with significant digits
  - transducer specifications
  - installation details such as physical location and ID of transducer

2.2.1 IEEE 1451.2 standard

The 1451.2 smart sensor standard defines a "plug-and-play" capability in a transducer module, which is achieved through an "electronic data sheet". It defines a digital interface to access this data sheet, read sensor data and set actuators. A set of read and write logic functions to access the "transducer electronic data sheet" and transducers are specified. The standard tries to reduce the difficulties designers have traditionally faced in establishing communications between various networks and transducers. The main goal of the standard is to provide an industry interface to efficiently connect transducers to microcontrollers and to connect microcontrollers to networks.

Sensor-networking is a fast growing technology. The transducer industry these days is dominated by analog transducers and interfacing these transducers to a measurement and control system is a hassle and very costly. Multiple control networking solutions are developing, each requiring a separate and significant effort on the part of transducer manufacturers.

However, networked transducers offer significant advantages to users, such as: introduction of self identification of sensors to the network or system, self-documentation, improved accuracy and reliability using digital communication, enhanced product functionality (e.g., diagnostics, multi-variable sensors, remote programmability), cost reductions in wiring and installation, in process design cycle, and in commissioning time. All these are significant contributions to productivity improvement and will benefit producers, vendors, system integrators, and users.

However, there is currently no defined common digital communication interface standard between transducers and network capable application processors (NCAPs). Each transducer manufacturer has to build their own interfaces. Currently transducer manufacturers cannot afford to support several control networks. A project designed as IEEE P1451.2, standard for a Smart Transducer Interface for Sensors and Actuators - Transducer to Microprocessor Communication Protocol and Transducer Electronics Data Sheet (TEDS) and its Formats, is to address these issues.

The objective of this standard is to simplify the development of networked transducers by defining the hardware and software blocks that do not depend on

individual control networks. This project has developed a standard hardware interface to connect transducers to network-capable application processors or microprocessors.

A Transducer Electronic Data Sheet and its data formats are specified to provide a standard way of describing transducers to measurement and control systems. The project specifies a communication protocol for the digital interface. This protocol is used to access the TEDS, read sensor measurements and send digital settings to actuators. The digital interface is usable by different types of sensors and actuators.

The NCAP mediates between the STIM and a digital network, and may provide local intelligence. The STIM communicates to the network transparently via the 'Transducer Independent Interface' that links it to the NCAP.

2.3 Smart transducer functional specification

2.3.1 Transducer channel types

The six channel types are as follows that the standard provides :

- Sensor
- Actuator
- Buffered sensor
- Data sequence sensor / Buffered data sequence sensor
- Event sequence sensor

a) Sensor

A sensor measures some physical parameter like physical, biological and or chemical on demand and returns digital data representing that parameter. For each

triggering event, a new data set must be sampled. The data set available to be read must be the result of the most recent trigger event. A sensor measures the state of the environment.

b) Actuator

An actuator is a transducer that accepts an electrical signal and converts it into a physical action. When a triggering occurs, the actuator state changes to match the data set most recently set to it. An actuator set the state of the environment.

c) Buffered sensor

A buffered sensor is different from a simple sensor because it has a single level of data buffering on the output channel. For each triggering event, a new data set must be sampled. The data set available to be read must be the data of the second most recent trigger event.

d) Data sequence sensor / Buffered data sequence sensor

A data sequence sensor acquires data continuously. The data set selected must be the one acquired immediately following the trigger. The data collection process in the must be enabled or disabled by means of a control command. The data set acquisition timing need not be periodic.

e) Event sequence sensor

An event sensor produces a signal whenever a specific event occurs. This signal must be the same signal that is used by sensors and actuators to acknowledge triggering

events. The event may be a digital signal transition or an analog signal crossing a setpoint.

2.4 STIM Functions

Each STIM shall implement the following functions:

- Addressing
- Interface data transport
- Meta-TEDS
- Global status
- Global control
- Triggering
- Hot-swap capability
- Interrupt
- Interrupt masking

Each channel of a STIM shall implement the following functions:

- Channel TEDS
- Transducer data
- Status
- Control

Each channel may also implement the following optional functions:

- Calibration TEDS
- Calibration Identification TEDS

- Channel Identification TEDS
- End-User's Application-Specific TEDS.
- Generic Extension TEDS
- Self calibration
- Self-test

2.5 Existing STIM

a) PICMicro MCU as an IEEE 1451.2 Compatible Smart Transducer Interface Module (STIM)

There is currently no defined independent digital communication interface between transducers and microprocessors. Each vendor builds its own interface. Without an independent, openly defined interface, transducer interfacing and integration are time consuming and duplicated efforts by vendors are economically unproductive. This interface provides a minimum implementation subset that allows self-identification and configuration of sensors and actuators, and also allows extensibility by vendors to provide growth and product differentiation.

The Hewlett-Packard BFOOT-66501 Embedded Ethernet Controller is the NCAP (Network Capable Application Processor) for this application. NCAP primarily mediates between the STIM and a particular interface. The NCAP may also perform correction of the raw data from the STIM and may include application specific data processing and control functionality. The controller is a complete thin web server solution for manufacturers of smart sensors and actuators, or for manufacturers of

products with embedded control who need a way to rapidly create smart Ethernet-ready devices that can scale to complete solutions. The BFOOT-66501 supports TCP/IP, FTP and HTTP protocols. The Microchip PIC16C62A and 93C86 EEPROM are the hardware base for the STIM in this project.

The physical connection between the NCAP and the PIC16C62A (STIM) is a 10-pin Transducer Independent Interface (TII). The TII is built around a synchronous serial communications based on the Serial Peripheral Interface (SPI) protocol. The SSP module of the PIC16C62A is used for SPI communications between the NCAP and STIM. When developing the TEDS, one parameter to consider is the maximum data rate for SPI transfers. The respective PICmicro MCU specification should also be considered for this TEDS entry. The 1451.2 interface provides for the sequence of reading and writing between the NCAP and the STIM. The top level protocols are read frame, write frame and triggering.

In this product, Channel 1 is a 16-bit actuator and Channel 2 is a 16-bit sensor. When the device is triggered, the cached actuator value is copied to the sensor data buffer. The actuator has a "divide by 10" and the sensor has a "multiply by 10" correction. With the conversion from float to unsigned short integer (uint16), the following loopback transformation occurs: 1) Write channel 1 with 123.4; 2) Read 120 from channel 2.

The goal of the IEEE 1451.2 standard is to provide an industry standard interface to efficiently connect transducers to microcontrollers and to connect microcontrollers to networks. Microchip Technology offers a large portfolio of smart microcontrollers that can be implemented as the STIM for compliance with the IEEE 1451.2 standard.

2.5.1 Comparison with existing STIM

One existing system of STIM is examined and compared with this project. The existing system is the PICmicro® MCU as an IEEE 1451.2 Compatible Smart Transducer Interface Module (STIM). The comparison table is as shown in Table 2.16.

	PICmicro® MCU	Proposed STIM simulator
Implementation	Full module (hardware and based)	TEDS module (software based only)
Implemented language	C programming	Java programming
Types of TEDS data blocks used	Meta-TEDS, Meta-ID TEDS, Channel TEDS, Channel-ID TEDS, Calibration TEDS.	Meta-TEDS and Channel TEDS.
Transducer	16-bit sensor, 16-bit actuator.	Virtual transducers (Temperature sensor (AD950), actuator (fan)).
Memory type	Flash memory	Flash memory

Table 2.0 : Comparison between existing system and proposed system

2.6 Chapter summary

The overview of the technical aspects of IEEE 1451.2 has been presented in this chapter. In addition, some of the practical considerations for implementation support of the standard have been discussed.

Chapter 3: Methodology

3.0 Chapter introduction

This chapter entails the entire methodology for building the TEDS simulator. The tools to build the TEDS will also be discussed. Methodology here means a way of developing a software product.

3.1 Object oriented approach

To program the TEDS, an object oriented programming will be approached. Object oriented programming is a type of programming in which it define not only the data type of data structures but also the type of operations (functions) that can be applied to the data structure. In this way, the data structure becomes an object that include both data and functions. In addition, relationship between one object and another can be created. For example, objects can inherit characteristics from another objects.

One of the principal advantages of object-oriented programming techniques over procedural programming techniques is that they enable programmers to create modules that do not need to be changed when a new type of object is added. A programmer can simply create a new object that inherits many of its features from existing objects. This makes object-oriented programs easier to modify.

The first step in OOP is to identify all the objects you want to manipulate and how they relate to each other, an exercise often known as data modeling. Once you've identified an object, you generalize it as a class of objects and define the kind of data it contains and any logic sequences that can manipulate it. Each distinct logic sequence is

known as a method. A real instance of a class is called (no surprise here) an "object" or, in some environments, an "instance of a class." The object or class instance is what you run in the computer. Its methods provide computer instructions and the class object characteristics provide relevant data. You communicate with objects - and they communicate with each other - with well-defined interfaces called messages.

The concepts and rules used in object-oriented programming provide these important benefits:

- The concept of a data class makes it possible to define subclasses of data objects that share some or all of the main class characteristics. Called inheritance, this property of OOP forces a more thorough data analysis, reduces development time, and ensures more accurate coding.
- Since a class defines only the data it needs to be concerned with, when an instance of that class (an object) is run, the code will not be able to accidentally access other program data. This characteristic of data hiding provides greater system security and avoids unintended data corruption.
- The definition of a class is reuseable not only by the program for which it is initially created but also by other object-oriented programs (and, for this reason, can be more easily distributed for use in networks).
- The concept of data classes allows a programmer to create any new data type that is not already defined in the language itself.

One of the first object-oriented computer languages was called Smalltalk. C++ and Java are the most popular object-oriented languages today. The Java programming language is designed especially for use in distributed applications on corporate networks and the Internet.

3.2 Project development tools

There are many programming languages that can be used to program the TEDS. Current existing STIM are built using C. In this project Java programming language would be used to build the TEDS. This is because Java implements the object oriented approach. Java is a simple, object-oriented, distributed, interpreted, robust, secure, architecture neutral, portable, multithreaded, and dynamic.

3.2.1 Java is simple

No language is simple, but Java considered a much simpler and easy to use object-oriented programming language when compared to the popular programming language, C++. Partially modeled after C++, Java has replaced the complexity of multiple inheritance in C++ with a simple structure called interface, and also has eliminated the use of pointers.

Java is Object-oriented programming models the real world. Everything in the world can be modeled as an object. For example, a circle is an object, a person is an object, and a window's icon is an object. Even a mortgage can be perceived as an object.

Java is object-oriented because programming in Java is centered on creating objects, manipulating objects, and making objects work together.

An object has properties and behaviors. Properties are described by using data, and behaviors are described by using methods. Objects are defined by using classes in Java. A class is like a template for the objects. An object is a concrete realization of a class description. The process of creating an object class is called instantiation. Java consists of one or more classes that are arranged in a treelike hierarchy, so that a child class is able to inherit properties and behaviors from its parent class. An extensive set of pre-defined classes, grouped in packages that can be used in programs are found in Java.

Object-oriented programming provides greater flexibility, modularity and reusability. For years, object-oriented technology has been perceived as an elitist, requiring substantial investments in training and infrastructure. Java has helped object-oriented technology enter the mainstream of computing, with its simple and clean structure that allows the programmer to write easy to read and write programs.

3.2.2 Java is distributed

Distributed computing involves several computers on a network working together. Java is designed to make distributed computing easy with the networking capability that is inherently integrated into it. Writing network programs in Java is like sending and receiving data to and from a file.

3.2.3 Portability: Program once, Run anywhere (Platform Independence)

One of the most compelling reasons to move to Java is its platform independence. Java runs on most major hardware and software platforms, including Windows 95 and NT, the Macintosh, and several varieties of UNIX. Java applets are supported by all Java-compatible browsers. By moving existing software to Java, you are able to make it instantly compatible with these software platforms. JAVA programs become more portable. Any hardware and operating system dependencies are removed.

3.2.4 Java is interpreted

An interpreter is needed in order to run Java programs. The programs are compiled into Java Virtual Machine code called bytecode. The bytecode is machine independent and is able to run on any machine that has a **Java** interpreter. Usually, a compiler will translate a high-level language program to machine code and the code is able to only run on the native machine. If the program is run on other machines, the program has to be recompiled on the native machine. With Java, the program need only be compiled once, and the bytecode generated by the Java compiler can run on any platform.

3.2.5 Security

Java is one of the first programming languages to consider security as part of its design. The Java language, compiler, interpreter, and runtime environment were each developed with security in mind. The compiler, interpreter, and Java-compatible browsers all contain several levels of security measures that are designed to reduce the

risk of security compromise, loss of data and program integrity, and damage to system users. Considering the enormous security problems associated with executing potentially untrusted code in a secure manner and across multiple execution environments, Java's security measures are far ahead of even those developed to secure military systems. C and C++ do not have any intrinsic security capabilities.

3.2.6 Reliability

Security and reliability go hand in hand. Security measures cannot be implemented with any degree of assurance without a reliable framework for program execution. Java provides multiple levels of reliability measures, beginning with the Java language itself. Many of the features of C and C++ that are detrimental to program reliability, such as pointers and automatic type conversion, are avoided in Java. The Java compiler provides several levels of additional checks to identify type mismatches and other inconsistencies. The Java runtime system duplicates many of the checks performed by the compiler and performs additional checks to verify that the executable bytecodes form a valid Java program.

3.2.7 Java is robust

Robust means reliable and no programming language can really assure reliability. Java puts a lot of emphasis on early checking for possible errors, as Java compilers are able to detect many problems that would first show up during execution time in other languages. Java eliminates certain types of programming constructs in other languages that are prone to errors. For instance, Java does not support pointers, which eliminates the possibility of overwriting memory and corrupting data. Java has a

runtime exception-handling feature to provide programming support for robustness, and can catch and respond to an exceptional situation so that the program can continue its normal execution and terminate gracefully when a runtime error occurs.

3.2.8 Java is portable

One advantage of Java is that its programs can run on any platform without having to be recompiled. This is one positive aspect of portability. It goes on even further to ensure that there are no platform-specific features on the Java language specification. For example, in some languages, such as Ada, the largest integer varies on different platforms. In Java, the size of the integer is the same on every platform, as is the behavior of arithmetic. Having a fixed size for numbers makes Java programs portable. The Java environment itself is portable to new hardware and operating systems, and in fact, the Java compiler itself is written in Java.

3.2.9 Java is multithreaded

Multithreaded is the capability for a program to perform several tasks simultaneously within a program. For instance, downloading a mp3 file while playing the file would be considered multithreading. In Java, multithreaded programming has been smoothly integrated into it, while in other languages, operating system-specific procedures have to be called in order to enable multithreading. Multithreading is especially useful in graphical user interface (GUI) and network programming. In GUI programming, many things can occur at the same time. For example, a user is able to listen to a mp3 file and surf the Web at the same time. In network programming, a server can serve multiple

clients at the same time. Multithreading is a necessity in visual and network programming.

3.3 Development tools

For this project the development tools used are as follows:

- Java 2 Platform, Standard Edition (J2SE 1.4.2)
- JCreator LE
- The system to be developed under Windows XP Professional

3.3.1 J2SE v1.4.2

The Java 2 Platform, Standard Edition is at the core of Java technology, and version 1.4.2 raises the java platform to a higher standard. It provides a complete environment for applications development on desktops and servers. It also primarily involve GUI, connectivity, virtual machine for the Java platform* (Java Virtual Machine (JVM)) and core libraries. From client to server, from desktop to supercomputer, improvement has been made to J2SE across the board. With version 1.4.2, enterprises can now use Java technology to develop more demanding business applications with less effort and in less time.

Version 1.4.2 builds upon the current J2SE platform and provides even more features for developers to build into their applications. More functionality in 1.4.2 means developers can now spend less time writing custom code to accomplish what is now part of the core J2SE platform. The result is faster applications programming with more

consistency for enterprise development initiatives. New features in J2SE v1.4.2 also reduce the developer's reliance on other technologies such as C or C++, PERL, SSL and DOM implementation in browsers. This allows developers to use a single technology to develop, test and deploy end-to- end enterprise application and software. Most anything you want to do, you can do it in J2SE user interface.

Version 1.4.2 provides more ways for developers leverage existing systems without changing their underlying platforms. It provides additional support for industry standards technologies such as XML, DOM, SSL, Kerberos, LDAP and CORBA to ensure operability across heterogeneous platforms, systems and environments. Additionally developers and software vendors may now use a new endorsed standard override mechanism in version 1.4.2 to provide newer versions of endorsed standard such as CORBA, as they become available.

3.3.1.1 Java Foundation Class (JFC)

JFC are a set of Java class libraries provided as part of the Java platform to support building graphics user interface and graphics functionality for Java technology-based client applications ("Java applications"). JFC includes an extensive set of technologies that enable developers to create a rich interactive user interface for client applications that can run not only on Microsoft Windows but also on other increasingly popular platform such Mac OSX and Linux.

Features of JFC :

- Abstract Window Toolkit (AWT): APIs that enable programs to integrate into

native window system, including APIs for Drag Drop.

- Java 2D: APIs to enable advanced 2D graphics, imaging, text and printing.
- Swing GUI components
- Accessibility: APIs and assitive technologies for ensuring an applications is accessible to users with disabilities and meets government requirements for accessibility.
- Internationalization: API JFC technologies include support for creating application that can interact with users around the world using the user's own language. This includes the Input Method Framework API.

These five technologies are designed to be used together to enable developers to build fully functional GUI client applications that run and integrate on any client machine that supports J2SE, including Microsoft Windows, Solaris, Linux and Mac OSX.

3.3.2 JCreator LE

JCreator is a powerful IDE (Integrated Development Environment) for Java technologies. It provides the user with templates, class browsers, a debugger interface, syntax highlighting, wizards and a fully customizable user interface.

User can directly compile or run their Java programming without having to activate the main document first. JCreator will automatically find the file with the main method or the html file holding the applet, and then start the appropriate tool

Users can also create their own tools for calling Java Development Kit (JDK) applications such as the complier, interpreter or application viewer. JCreator also

supports multiple complier tools that can be switched with the runtime configuration dialogue box.

3.4 Chapter summary

In this chapter, the design methodology and the development tools have been discussed. Java has significant advantages not only as a commercial language but also as a teaching language. It allows students to learn object-oriented programming without exposing them to the complexity of C++. It provides the kind of rigorous compile-time error checking typically associated with Pascal.

Chapter 4 : System Analysis

4.0 Chapter introduction

In this chapter all the requirements and consideration are analyzed and translated into a software model. All the specification needed to build the simulator will be discussed.

4.1 Transducer Electronic Data Sheet (TEDS) specification

There are 8 types of TEDS data block, that is Meta-TEDS, Channel TEDS, Calibration TEDS, Meta-Identification TEDS, Channel Identification TEDS, Calibration Identification TEDS, End-Users' Application-Specific TEDS, and Generic Extension TEDS. Only the Meta-TEDS and the Channel TEDS are mandatory while the rest are optional.

All fields in the mandated TEDS data blocks shall be filled, unless a length field that applies to them is zero. If a field is not applicable to the implementation, its value shall be

- A null string for string data types
- A NaN for single-precision real and double-precision real data types
- A zero for integers, enumeration, and field length data types
- "Digital data", for physical unit data types

The following TEDS data blocks may have their length field set to zero:

- Meta-Identification TEDS
- Channel-Identification TEDS

- Calibration-Identification TEDS
- Calibration TEDS
- End-Users' Application-Specific TEDS
- Industry Extension TEDS

A TEDS length field of $FFFFFFFF_{16}$ shall be interpreted as a length of zero.

4.1.1 Functional requirements

Functional requirements specifies a function that a system or a system's component must be able to perform. These are software requirements that defines the behaviors of a system, that is the fundamental process of transformation that software and hardware components of the system perform on inputs to provide expected output. Below are the information that TEDS must keep.

4.1.1.1 Meta-TEDS data block

Function

The Meta-TEDS provide global information to the client about the STIM unit attached to the NCAP and include the information users need to access data in channel TEDS. The meta-TEDS provide common information applicable to all transducers attached to the STIM and include unique identifiers for the STIM, the number of transducers attached to the STIM, the longest delay time between a request for transducer data and

the delivery of the data for any of the transducers attached to the STIM, and the presence of a correction/compensation engine for sensors attached to the unit.

Access

Functional address 160 applied to CHANNEL_ZERO shall access this data.

Data structure

Refer to Table 4.0 to see the data structure. Serial transmission of data shall occur msb first. When serial data is divided into bytes, such as in the transmission of multi-byte TEDS fields, the most significant byte shall be transmitted first.

Field no.	Description	Type	No. of bytes
TEDS version constant related data sub-block			
1	Meta-TEDS Length	U32L	4
2	IEEE 1451 Standards Family Working Group Number	U8E	1
3	TEDS Version Number	U8E	1
Identification related data sub-block			
4	Globally Unique Identifier	UUID	10
Data structure related data sub-block			
5	CHANNEL_ZERO Industry Calibration TEDS Extension Key	U8E	1
6	CHANNEL_ZERO Industry Nonvolatile Data Fields Extension Key	U8E	1
7	CHANNEL_ZERO industry TEDS extension key	U8E	1
8	CHANNEL_ZERO End-Users' Application-Specific TEDS key	U8E	1
9	Number of Implemented Channels	U8C	1
10	Worst-Case Channel Data Model Length	U8C	1
11	Worst-Case Channel Data Repetitions	U16C	2
12	CHANNEL_ZERO writable TEDS length	U32C	4
Timing related data sub-block			
13	Worst-Case Channel Update Time (t_{wu})	F32	4
14	Global Write Setup Time (t_{gws})	F32	4
15	Global Read Setup Time (t_{grs})	F32	4
16	Worst-Case Channel Sampling Period (t_{wsp})	F32	4

17	Worst-Case Channel Warm-Up Time	F32	4
18	Command Response Time	F32	4
19	STIM Handshake Timing (t_{hs})	F32	4
20	End-Of-Frame Detection Latency (t_{lat})	F32	4
21	TEDS Hold-Off Time (t_{th})	F32	4
22	Operational Hold-Off Time (t_{oh})	F32	4
23	Maximum Data Rate	U32C	4
Channel grouping related data sub-block			
24	Channel Groupings Data Sub-block Length	U16L	2
25	Number of Channel Groupings = G	U8C	1
Fields 26Ð28 are repeated G times once for each group			
26	Group Type U8E	1	
27	Number of Group Members = N	U8C	1
28	Member Channel Numbers List = M(N)	Array of U8E	N
Data integrity data sub-block			
29	Checksum for Meta-TEDS	U16C	2

Table 4.0 : Data structure of Meta-TEDS data block

Meta-TEDS Length

Meta-TEDS data field number 1

Data type: unsigned integer used for field length (U32L, 4 bytes)

It specifies the total number of bytes in the Meta-TEDS data block excluding this field.

IEEE 1451 Standards Family Working Group Number

Meta-TEDS data field number 2

Data type: unsigned byte integer used for enumeration (U8E, 1 byte)

It must be set to two for devices conforming to this standard. (IEEE 1451.2)

TEDS Version Number

Meta-TEDS data field number 3

Data type: unsigned byte integer used for enumeration (U8E, 1 byte)

It defines the version number of the TEDS that corresponds to the particular IEEE 1451 standard of the working group that specifies the TEDS data structure as shown in Table 4.1.

TEDS version	IEEE 1451.2 standard version
0	Reserved
1	This will correspond to the first official version of the standard: IEEE Std 1451.2-1997.
2-255	Reserved

Table 4.1 : Enumeration of TEDS Version Numbers

Globally Unique Identifier

Meta-Identification TEDS data field number 4

Data type: Universally unique identification (UUID, 10 bytes)

This field is provided to allow better management of STIM components in a distributed system (e.g., tracking and traceability of STIMs for operational and maintenance purposes) and must be guaranteed to be unique in the universe of all STIMs.

CHANNEL_ZERO Industry Calibration TEDS Extension Key

Meta-TEDS data field number 5

Data type: unsigned byte integer used for enumeration (U8E, 1 byte)

The value in this field specifies the highest functional address for writing the industry-implemented Calibration TEDS extension that is available in the STIM for CHANNEL_ZERO. Acceptable values and their meanings are defined in Table 4.2.

Key value (K)	Meaning
0	No extensions implemented in STIM
1-79	Invalid
80-95	Valid TEDS extension(s) implemented for: -Functional addresses used for writing: between 80 and (K); and -Functional addresses used for reading: between 208 and (K+128)
96-255	Invalid

Table 4.2 : Enumeration of CHANNEL_ZERO Industry Calibration TEDS Extension Keys

CHANNEL_ZERO Industry Nonvolatile Data Fields Extension Key

Meta-TEDS data field number 6

Data type: unsigned byte integer used for enumeration (U8E, 1 byte)

The value in this field specifies the highest functional address for writing the industry-implemented nonvolatile data field extensions that is available in the STIM for CHANNEL_ZERO. Acceptable values and their meanings are defined in Table 4.3.

Key value (K)	Meaning
0	No extensions implemented in STIM
1-111	Invalid
112-127	Valid TEDS extension(s) implemented for: -Functional addresses used for writing: between 112 and (K); and -Functional addresses used for reading: between 240 and (K+128)
128-255	Invalid

Table 4.3 : Enumeration of CHANNEL_ZERO Industry Nonvolatile Data Fields Extension Keys

CHANNEL_ZERO Industry TEDS Extension Key

Meta-TEDS data field number 7

Data type: unsigned byte integer used for enumeration (U8E, 1 byte)

The value in this field specifies the highest functional address for writing the industry-implemented TEDS extension that is available in the STIM for CHANNEL_ZERO. Acceptable values and their meanings are defined in Table 4.4.

Key value (K),	**Meaning**
0	No extensions implemented in STIM
1-175	Invalid
176-191	Valid TEDS extension(s) implemented for: - Functional addresses used for reading: between 176 and (K)
192-255	Invalid

Table 4.4 : Enumeration of CHANNEL_ZERO Industry TEDS Extension Keys

CHANNEL_ZERO End-Users' Application-Specific TEDS Key

Meta-TEDS data field number 8

Data type: unsigned byte integer used for enumeration (U8E, 1 byte)

This field specifies the presence of End-Users' Application-Specific TEDS function in CHANNEL_ZERO as defined in Table 4.5.

Key value	**Meaning**
0	End-Users' Application-Specific TEDS is not implemented on CHANNEL_ZERO.
1	End-Users' Application-Specific TEDS is implemented on CHANNEL_ZERO.
2-255	Reserved.

Table 4.5 : Enumeration of End-Users' Application-Specific TEDS Keys

Number of Implemented Channels

Meta-TEDS data field number 9

Data type: unsigned byte integer used for counting (U8C, I byte)

This field indicates the number of channels implemented in the STIM.

Worst-Case Channel Data Model Length

Meta-TEDS data field number 10

Data type: unsigned byte integer used for counting (U8C, 1 byte)

This field specifies the maximum value of Channel Data Model Length for all the implemented channels.

Worst-Case Channel Data Repetitions

Meta-TEDS data field number 11

Data type: unsigned 16 bit integer used for counting (U16C, 2 bytes)

This field specifies the length in bytes available for all the implemented channels.

CHANNEL_ZERO Writable TEDS length

Meta-TEDS data field number 12

Data type: unsigned 32 bit integer used for counting (U32C, 4 bytes)

This field specifies the length in bytes available for each CHANNEL_ZERO user-writable TEDS. The only structure currently defined in the standard is the CHANNEL_ZERO End-Users' Application-Specific TEDS.

Worst-Case Channel Update Time

Meta-TEDS data field number 13

Data type: single-precision real (F32, 4 bytes)

This field specifies the maximum value of the Channel Update Time (t_{wu}) for all the implemented channels in seconds.

Global Write Setup Time

Meta-TEDS data field number 14

Data type: single-precision real (F32, 4 bytes)

This field defines the minimum time (t_{gws}), in seconds, between the end of a global write frame and the application of a global trigger.

Global Read Setup Time

Meta-TEDS data field number 15

Data type: single-precision real (F32, 4 bytes)

This field defines the minimum time (t_{grs}), in seconds, between the receipt of a global trigger acknowledge and the beginning of a global read frame.

Worst-Case Channel Sampling Period

Meta-TEDS data field number 16

Data type: single-precision real (F32, 4 bytes)

This field indicates the maximum value (t_{wsp}), in seconds, of the channel sampling period for all implemented channels.

Worst-Case Channel Warm-Up Time

Meta-TEDS data field number 17

Data type: single-precision real (F32, 4 bytes)

This field indicates the minimum time, in seconds, that is necessary between application of power to the STIM and activation of the first transducer data transfer. This is the maximum value of all the Channel Warm-Up Times.

Command Response Time

Meta-TEDS data field number 18

Data type: single-precision real (F32, 4 bytes)

This field indicates the longest time, in seconds, that the STIM takes to process any commands (read/write).

STIM Handshake Time

Meta-TEDS data field number 19

Data type: single-precision real (F32, 4 bytes)

This field indicates the longest time (t_{hs}), in seconds, for the STIM to remove the trigger acknowledge signal after the signal is removed by the NCAP, or for the STIM to remove the data transport acknowledge signal after the data transport is inactivated by the NCAP.

End-Of-Frame Detection Latency

Meta-TEDS data field number 20

Data type: single-precision real (F32, 4 bytes)

This field indicates the longest time (t_{lat}), in seconds, that a STIM must take to detect the removal of the data transport enable signal., The STIM must be ready to detect another assertion of the data transport enable signal, which the STIM must stand to be the start of new data transport frame, if the data transport enable signal is removed for this period or longer

TEDS Hold-Off Time

Meta-TEDS data field number 21

Data type: single-precision real (F32, 4 bytes)

This field defines the maximum individual hold-off time, in seconds, done by the STIM before the first byte, or between bytes, of any data transfer addressed to TEDS functions, (i.e., functional addresses in the ranges of 32-127 or 160-255, inclusive).

Operational Hold-Off Time

Meta-TEDS data field number 22

Data type: single-precision real (F32, 4 bytes)

This field defines the maximum individual hold-off time, in seconds, done by the STIM before the first byte, or between bytes, of any data transfer addresses to operational functions, (i.e., functional addresses in the ranges of 1-32 or 129-159, inclusive).

Maximum Data Rate

Meta-TEDS data field number 23

Data type: unsigned 32 bit integer used for counting (U32C, 4 bytes)

This field indicates the maximum data rate, in bits per second, supported by the STIM interface.

Channel Groupings Data Sub-Block Length

Meta-TEDS data field number 24

Data type: unsigned 16 bit integer used for field length (U16L, 2 bytes)

This field defines the total number of bytes in the Channel Grouping data sub-block. The Channel Groupings Data Sub-Block Length field must not include the length of the length field itself. If the value is zero, the Channel Groupings is not defined.

Number of Channel Groupings

Meta-TEDS data field number 25

Data type: unsigned byte integer used for counting (U8C, 1 byte)

This field indicates the number of discrete channel groupings defined in this STIM's Meta-TEDS.

Group Type

Meta-TEDS data field number 26

Data type: unsigned byte integer used for enumeration (U8E, 1 byte)

The relationship between the channels comprising the specific group shall be defined by the enumeration in Table 4.6.

Value	Meaning
0	An arbitrary relation
1	x, y, z right-hand rectangular spatial coordinates
2	ρ, φ, z right-hand cylindrical spatial coordinates

3	r, θ ,Φ right-hand spherical spatial coordinates
4	Latitude longitude altitude planetary coordinates
5	In-phase quadrature temporal coordinates
6	Red, green, blue color coordinates
7	Analog event sequence sensor channel, analog input sensor channel, upper threshold virtual actuator channel hysteresis virtual actuator channel
8	Sensor channel (any type) high-pass filter virtual actuator channel low-pass filter virtual actuator channel scale factor virtual actuator channel
9	Transducer (any type) sample interval virtual actuator channel
10	Digital event sequence sensor channel , digital input sensor channel, event pattern virtual actuator channel
11-127	Reserved for future expansion
128-255	Open to industry

Table 4.6 : Enumeration of Group Types

Number of Group Members

Meta-TEDS data field number 27

Data type: unsigned byte integer used for counting (U8C, 1 byte)

This field specifies the number of channels comprising the specific group.

Member Channel Numbers List

Meta-TEDS data field number 28

Data type: an array of unsigned byte integers used for enumeration (U8E, 0 to 255 bytes)

This field indicates a one-dimensional array (list) of 1 byte elements. Each element is the channel address for a member channel in the specific group.

Checksum for Meta-TEDS

Meta-TEDS data field number 29

Data type: unsigned 16 bit integer used for counting (U16C, 2 bytes)

It contains the checksum for the complete Meta-TEDS data block. The checksum shall be the one's complement of the sum (modulo 216) of all the data structure's preceding bytes, including the initial length field and excluding the checksum field.

4.1.1.2 Channel-TEDS data block

Function

The Channel TEDS contain information specific to one of the transducers connected to the STIM; each sensor and each actuator connected to the STIM must have its own channel TEDS. The type of information in the channel TEDS includes the physical units associated with the data, acquisition delays, the upper and lower limits on the data, and the type of data—such as single precision real, 4-byte integer.

Access

Functional address 160 applied to channels 1-255 shall access this data.

Data Structure

Refer to Table 4.7 to see the data structure. Serial transmission of data shall occur msb first. When serial data is divided into bytes, such as in the transmission of multi-byte TEDS fields, the most significant byte shall be transmitted first.

Field No.	Description	Type	No. of bytes
Data structure related data sub-block			

1	Channel TEDS Length	U32L	4
2	Calibration Key	U8E	1
3	Channel Industry Calibration TEDS Extension Key	U8E	1
4	Channel Industry Nonvolatile Data Fields Extension Key	U8E	1
5	Channel Industry TEDS Extension Key	U8E	1
6	Channel End-Users' Application-Specific TEDS Key	U8E	1
7	Channel Writable TEDS Length	U32C	4
Transducer related data sub-block			
8	Channel Type Key	U8E	1
9	Physical Units	UNITS	10
10	Lower Range Limit	F32	4
11	Upper Range Limit	F32	4
12	Worst-Case Uncertainty	F32	4
13	Self-Test Key	U8E	1
Data converter related data sub-block			
14	Channel Data Model	U8E	1
15	Channel Data Model Length	U8C	1
16	Channel Model Significant Bits	U16C	2
17	Channel Data Repetitions	U16C	2
18	Series Origin	F32	4
19	Series Increment	F32	4
20	Series Units	UNITS	10
Timing related data sub-block			
21	Channel Update Time (t_u)	F32	4
22	Channel Write Setup Time (t_{ws})	F32	4
23	Channel Read Setup Time (t_{rs})	F32	4
24	Channel Sampling Period (t_{sp})	F32	4
25	Channel Warm-Up Time	F32	4
26	Channel Aggregated Hold-Off Time (t_{ch})	F32	4
27	Timing Correction	F32	4
28	Trigger Accuracy	F32	4
Event sequence options field			
29	Event Sequence Options	U8E	1
Data integrity data sub-block			
30	Checksum for Channel TEDS	U16C	2

Table 4.7 : Data structure of Channel TEDS data block

Channel TEDS Length

Channel TEDS data field number 1

Data type: unsigned 32 bit integer used for counting (U32L, 4 bytes)

It indicates the total number of bytes in the channel TEDS data block excluding this field.

Calibration Key

Channel TEDS data field number 2

Data type: unsigned byte integer used for enumeration (U8E, 1 byte)

The calibration capabilities of the STIM are defined in Table 4.8.

Value	Name	Function
0	CAL_NONE	No calibration information needed or provided. No correction is performed by the NCAP on transducer data associated with this channel. If the value is CAL_NONE this implies that there is no calibration TEDS associated with this block. If the Calibration TEDS is accessed the Calibration TEDS Length shall be zero.
1	CAL_FIXED	Fixed calibration information provided. This information cannot be modified. Correction is performed in the NCAP or elsewhere in the system.
2	CAL_MODIFIABLE	Calibration information provided. This information can be modified by writing to the Calibration TEDS. Correction is performed in the NCAP or elsewhere in the system.
3	CAL_SELF	Calibration information provided. Adjusted by a self-calibration capability. Correction is performed in the NCAP or elsewhere in the system.

4	CAL_CUSTOM	Calibration information is provided through an industry extension. Correction is performed in the NCAP or elsewhere in the system.
5	STIM_CAL_FIXED	Fixed calibration information is provided to be applied in the STIM. This information cannot be modified.
6	STIM_CAL_MODIFIABLE	Calibration information is provided to be applied in the STIM. This information can be modified by writing to the Calibration TEDS.
7	STIM_CAL_SELF	Calibration information is provided to be applied in the STIM. Adjusted by a self-calibration capability.
8-255	Reserved	Reserved for future expansion.

Table 4.8 : Enumeration of Calibration Keys

NCAP Correction

Calibration key enumerations CAL_FIXED, CAL_MODIFIABLE, CAL_SELF, and CAL_CUSTOM are to be used when the correction is performed in the NCAP or elsewhere in the system.

STIM Correction

Calibration key enumeration STIM_CAL_FIXED, STIM_CAL_MODIFIABLE, and STIM_CAL_SELF are to be used when the correction is performed in the STIM.

Channel Industry Calibration TEDS Extension Key

Channel TEDS data field number 3

Data type: unsigned byte integer for enumeration (U8E, 1 byte)

The value in here specifies the highest functional address for writing the industry-implemented Calibration TEDS extension that is available in the STIM for this channel. Acceptable values and their meanings are defined in Table 4.9.

Key value (K)	Meaning
0	No extensions implemented in STIM
1-79	Invalid
80-95	Valid TEDS extension(s) implemented for: - Functional addresses used for writing: between 80 and (K); and - Functional addresses used for reading: between 208 and (K+128)
96-255	Invalid

Table 4.9 : Enumerations of Channel Industry Calibration TEDS Extension Keys

Channel Industry Nonvolatile Data Fields Extension Key

Channel TEDS data field number 4

Data type: unsigned byte integer used for enumeration (U8E, 1 byte)

The value in here specifies the highest functional address for writing the industry-implemented nonvolatile data field extensions that is available in the STIM for this channel. Acceptable values and their meanings are defined in Table 4.10.

Key value (K)	Meaning
0	No extensions implemented in STIM
1-111	Invalid
112-127	Valid TEDS extension(s) implemented for: - Functional addresses used for writing: between 112 and (K); and - Functional addresses used for reading: between

	240 and (K+128)
128-255	Invalid

Table 4.10 : Enumerations of Channel Industry Nonvolatile Data Fields Extension Keys

Channel Industry TEDS Extension Key

Channel TEDS data field number 5

Data type: unsigned byte integer used for enumeration (U8E, 1 byte)

The value in here specifies the highest functional address for writing the industry-implemented TEDS extensions that is available in the STIM for this channel. Acceptable values and their meanings are defined in Table 4.11.

Key value (K)	**Meaning**
0	No extensions implemented in STIM
1-175	Invalid
176-191	Valid, TEDS extension(s) implemented for: - Functional addresses used for reading: between 176 and (K)
192-255	Invalid

Table 4.11 : Enumerations of Channel Industry TEDS Extension Keys

Channel End-Users' Application-Specific TEDS Key

Channel TEDS data field number 6

Data type: unsigned byte integer used for enumeration (U8E, 1 byte)

This field specifies the presence of End-Users' Application-Specific TEDS function for this channel as defined in Table 4.12.

Key value	Meaning
0	End-Users' Application-Specific TEDS Function Is Not Implemented On This Channel.
1	End-Users' Application-Specific TEDS function is implemented on this channel.
2-255	Reserved

Table 4.12 : Enumeration of End-Users' Application-Specific TEDS Keys

Channel Writable TEDS Length

Channel TEDS data field number 7

Data type: unsigned 32 bit integer used for counting (U32C, 4 bytes)

This field indicates the length in bytes available for each individual user-writable TEDS associated with this channel, such as Calibration TEDS, Calibration Identification TEDS, or End-Users' Application-Specific TEDS. An entire writable TEDS, including the length-field and checksum, must fit within this maximum length.

Channel Type Key

Channel TEDS data field number 8

Data type: unsigned byte integer used for enumeration (U8E, 1 byte)

This field indicates the channel transducer type. The values for Channel Type Key are defined in Table 4.13.

Key value	Meaning
0	Sensor
1	Actuator
2	Event sequence sensor

3	Data sequence sensor
4	General transducer
5	Buffered sensor
6	Buffered data sequence sensor
7-255	Reserved for future expansion

Table 4.13 : Enumeration of Channel Type Keys

Physical Units

Channel TEDS data field number 9

Data type: Physical units (UNITS, 10 bytes)

This field specifies the physical units that apply to the transducer data, however, if the Calibration Key is CAL_FIXED, CAL_MODIFIABLE, CAL_SELF, or CAL_CUSTOM the physical units apply only to the transducer data *after* correction for the case of sensors, or *before* correction for the case of actuators.

Lower Range Limit

Channel TEDS data field number 10

Data type: single-precision real (F32, 4 bytes)

For sensors, this must be the lowest valid value for transducer data after correction is done, changed in the units specified by the Physical Units field of the Channel TEDS. If the corrected transducer data lies below this limit, it may not be compatible with STIM specifications set by the manufacturer.

For actuators, this shall be the lowest valid value for transducer data before correction is done, changed in the units specified by the physical units field of the channel TEDS.

Writing corrected transducer data below this limit, may not be compatible with STIM specifications set by the manufacturer.

Upper Range Limit

Channel TEDS data field number 11

Data type: single-precision real (F32, 4 bytes)

For sensors, this shall be the highest valid value for transducer data after correction is done, changed in the units specified by the Physical Units field of the Channel TEDS. If the corrected transducer data lies above this limit, it may not be compatible with STIM specifications set by the manufacturer.

For actuators, this shall be the highest valid value for transducer data before correction is done, changed in the units specified by the Physical Units field of the Channel TEDS. Writing corrected transducer data above this limit may not be compatible with STIM specifications set by the manufacturer.

Worst-Case Uncertainty

Channel TEDS data field number 12

Data type: single-precision real (F32, 4 bytes)

This field indicates the "Combined Standard Uncertainty". The value of this field must be expressed in the same units as the transducer data as specified in the Physical Units field of the Channel TEDS.

Self-Test Key

Channel TEDS data field number 13

Data type: unsigned byte integer used for enumeration (U8E, 1 byte)

This field specifies the self-test capabilities of the transducer as shown in Table 4.14.

Key value	Meaning
0	No self-test function needed or provided
1	Self-test function provided
2-255	Reserved for future expansion

Table 4.14 : Enumeration of Self-Test Keys

Channel Data Model

Channel TEDS data field number 14

Data type: unsigned byte integer used for enumeration (U8E, 1 byte)

This field defines the data model used when addressing read transducer data or write transducer data for this channel as shown in Table 4.15.

There are two differences between N-byte integer (enumeration zero) and N-byte-fraction (enumeration three), as follows:

a) The radix point (which divides integer from fractional bits) is to the right of the lsb for N-byte-integer. It is immediately to the right of the msb for N-byte-fraction.

b) Justification of the significant bits differs.

Value	Model	Length
0	N-byte integer (unsigned)	0≤N≤255
1	Single-precision real	4 bytes
2	Double-precision real	8 bytes
3	N-byte fraction (unsigned)	0≤N≤255
4-255	Reserved for future expansion	-

Table 4.15 : Enumeration of Channel Data Models

The N-byte fraction type may be used to keep the multinomial coefficients within representable bounds.

Channel Data Model Length

Channel TEDS data field number 15

Data type: unsigned byte integer used for counting (U8C, 1 byte)

This field identifies the number of bytes in the representation of the selected Channel Data Model.

For N-byte integer	the value in this field shall be N, where $0 \leq N \leq 255$.
For N-byte fraction	the value in this field shall be N, where $0 \leq N \leq 255$.
For single-precision real	the value in this field shall be 4.
For double precision real	the value of this field shall be 8.

Channel Model Significant Bits

Channel TEDS data field number 16

Data type: unsigned 16 bit integer used for counting (U16C, 2 bytes)

When the Channel Data Model is N-byte integer (enumeration zero) or N-byte fraction (enumeration three), the value of this field is the number of bits that are significant. The value of this field must be between zero and 2040.

For example, if data from a transducer channel comes from a 12 bit ADC, then

Channel Data Model = N-byte integer (field enumeration value of zero)

Channel Data Model Length = 2 (the number of bytes to hold 12 bits)

Channel Model Significant Bits = 12

When the Channel Data Model is N-byte integer or N-byte fraction, the Channel Model Significant Bits must not exceed eight times the Channel Model Data Length.

When the Channel Data Model is N-byte integer, the significant data bits shall be right-justified within the byte stream.

When the Channel Data Model is N-byte fraction, the significant data bits must be left-justified within the byte stream.

When the Channel Data Model is single- or double-precision real (enumeration one or two), the value of this field is the number of bits in the STIM's signal converter.

Channel Data Repetitions

Channel TEDS data field number 17

Data type: unsigned byte integer used for counting (U16C, 2 bytes)

The number L of repetitions of the transducer value produced or required by a single trigger. The purpose of this structure must be to enable the specification of transducers that produce an array of data with the application of a single trigger such as a time series or mass spectrum.

Series Origin

Channel TEDS data field number 18

Data type: single-precision real (F32, 4 bytes)

For the case where the Channel Data Repetitions is bigger than zero, the Series Origin represents the value of the independent variable associated with the first datum returned in a data set.

Series Increment

Channel TEDS data field number 19

Data type: single-precision real (F32, 4 bytes)

For the cases where the Channel Data Repetitions is bigger than zero, the Series Increment represents the spacing between values of the independent variable associated with successive members of the data set.

Series Units

Channel TEDS data field number 20

Data type: Physical units (UNITS, 10 bytes)

This field specifies the physical units associated with the series origin and series increment fields in the Channel TEDS.

Channel Update Time

Channel TEDS data field number 21

Data type: single-precision real (F32, 4 bytes)

This field defines the maximum time (t_u), in seconds, between the receipt of a trigger and the issue of trigger acknowledge for this channel.

Channel Write Setup Time

Channel TEDS data field number 22

Data type: single-precision real (F32, 4 bytes)

This field defines the minimum time (t_{ws}), in seconds, between the end of a write frame and the application of a trigger.

Channel Read Setup Time

Channel TEDS data field number 23

Data type: single-precision real (F32, 4 bytes)

This field indicates the minimum time (t_{rs}), in seconds, between the trigger acknowledge and the beginning of a read frame.

Channel Sampling Period

Channel TEDS data field number 24

Data type: single-precision real (F32, 4 bytes)

The Channel Sampling Period (tsp) shall be the minimum sampling period of the channel transducer unencumbered by read or write considerations.

For sensor, buffered sensor, and actuator channels this time will be limited by A/D or D/A conversion times, STIM processor speed, etc., but in more complex transducers it may specify transducer or sample handling times as well.

The Channel Sampling Period shall be expressed in seconds.

Channel Warm-Up Time

Channel TEDS data field number 25

Data type: single-precision real (F32, 4 bytes)

This field indicates the period of time, in seconds, in which the device stabilizes its performance to predefined tolerances after the application of power to the transducer.

Channel Aggregated Hold-Off Time

Channel TEDS data field number 26

Data type: single-precision real (F32, 4 bytes)

This field indicates the maximum aggregated time (t_{ch}) that the STIM will spend holding off the data transducer during a complete data transfer addressed to *read transducer data* or *write transducer data* and this channel, assuming the Maximum Data Rate is used.

Timing Correction

Channel TEDS data field number 27

Data type: single-precision real (F32, 4 bytes)

This field defines the time offset, in seconds, between the issue of global trigger acknowledge and when this channel actually sampled the sensor or updated the actuator. If the channel itself is addressed, then the trigger acknowledge defines the actuation or sensing point in time, and the timing correction field does not apply.

Trigger Accuracy

Channel TEDS data field number 28

Data type: single-precision real (F32, 4 bytes)

This field defines the accuracy, in seconds, of the Timing Correction.

Event Sequence Options

Channel TEDS data field number 29

Data type: unsigned byte integer used for enumeration (U8E, 1 byte)

An event sequence sensor has the option of changeable pattern, upper threshold, and/or hysteresis and at the same time detecting inconsistencies in settings of these parameters. This enumeration defines, for the NCAP, the ability of the STIM to detect and report these inconsistencies. The options are defined in Table 4.16.

Value	Meaning
0	Not applicable
1	Pattern/threshold/hysteresis not changeable
2	Changeable and inconsistencies detected
3	Changeable and inconsistencies not detected
4-255	Reserved

Table 4.16 : Event sequence options

Checksum for Channel TEDS

Channel TEDS data field number 30

Data type: unsigned 16 bit integer used for counting (U16C, 2 bytes)

This field contains the checksum for the complete Channel TEDS data block. The checksum shall be the one's complement of the sum (modulo 216) of all the data structure's preceding bytes, including the initial length field and excluding the checksum field.

4.2 Sensors and actuators

For the purpose of this project, virtual sensor and actuators will be used as there is no any physical connection with the outside world for this simulator

4.2.1 Virtual sensor

A sensor is a component that measures the state of the environment. A logical sensor is a functional component that measures a parameter of the environment and provides the measure when requested. Abstracting away the physical interaction with the environment, sensors can be modeled as data sources with no input signals carrying data. A physical sensor, on the other hand, represents a hardware component with well-defined characteristics in terms of accuracy, stability, range, etc. Simple sensors are devices that directly map a physical quantity into an electrical signal and provide the measure upon request. Devices that measure physical parameters, such as temperature, sound, and light, as well as input devices such as keyboards or microphones that allow external users to enter data or set parameters are examples of simple sensors.

Virtual sensors are components that overall look like sensors in the sense that they provide data upon an external request. Virtual sensors are defined by the list of parameters that can be read and by the primitives that are used for reading them. Examples of virtual sensors is a sensor that provides an indirect measure of a certain environment condition by combining one or more sensing functions with processing (e.g., transformation, compression, aggregation).

4.2.2 Virtual actuators

An actuator is a component that sets the state of the environment. A logical actuator is a functional component that sets a parameter of the environment. Abstracting away the physical interaction with the environment, actuators can be modeled as data sinks with no output signals carrying data. A physical actuator, on the other hand, represents an actual hardware component with will-defined characteristics in terms of settings, accuracy, etc. Simple actuators that are devices that map an electrical signal into a physical quantity and may return an acknowledgment when the action is taken. Examples of actuators are devices that modify physical parameters, such as heaters and automatic window/door openers, as well as output devices, such as displays and speakers.

Virtual actuators that overall look like actuators in the sense that they receive values to set some parameters. Virtual actuators are defined by the list of parameters that can be set and by the primitives that are used for setting. Examples of virtual actuators are an actuator that provides an indirect way of controlling a certain environment condition by combining one or more physical actuators with processing (e.g., transformation, and decompression).

4.3 Non-Functional Requirements

(i) User interface

The TEDS system will be a stand alone system using a Java interface. The user interface will be simple for the ease of users.

(ii) Usability

The TEDS is designed as an emulator for researches and students in the areas of smart sensor. This project serves as a testbench for students who wants to see how the simulator works and apply it to real environment. It gives an opportunity to those who wants to enhance and implement the system.

(iii) Performance

The main idea of building the TEDS is to see it's behavior in test environment. We can also see how TEDS is written in Flash and how the data is encoded in the Flash.

(iv) Maintainability

System maintenance always require more effort and time if the system is not well planned and designed at the beginning. System maintenance is a must for this system, just like any other system as it allows certain changes or modifications to be made over the system. Some changes include the adding on more TEDS data block like Calibration TEDS.

4.4 Minimal hardware requirements

The following hardware will be utilized for the TEDS:

- 128Mbytes RAM
- AMD Athlon ™ Processor, 654 Mhz
- 12.66 GB hard disk drive via IDE interface
- 32.0 MB video cards

- PS/2 keyboard
- PS/3 trackball mouse

4.5 Software requirements

- As this projects going to be built using Java, the Java plug-in must be installed in order to view the interface. This software can be run in any platform as Java is platform independence.
- Java 2 Platform, Standard Edition (J2SE v1.4.2)
- JCreator LE
- Windows XP Professional

4.6 Data type

The data types used in TEDS are showed in Table 4.17.

Type	Usage For	Symbol	Size (byte(s))
Unsigned byte integer	Counting	U8C	1
Unsigned byte integer	Enumeration	U8E	1
Unsigned byte integer	Field Length	U8L	1
Unsigned 16 bit integer	Counting	U16C	2
Unsigned 16 bit integer	Enumeration	U16E	2
Unsigned 16 bit integer	Field Length	U16L	2
Unsigned 32 bit integer	Counting	U32C	4
Unsigned 32 bit integer	Field Length	U32L	4
Single-precision real	-	F32	4
Double precision real	-	F64	8
String	-	STRING	Variable
String language specification	-	LANG	3
String character set	Enumeration	U8E	1
Character code format	Enumeration	U8E	1
String Language Code	Enumeration	U8E	1
Physical units	-	UNITS	10
Universal unique identification	-	UUID	10

Table 4.17 : Data types

4.7 Memory used for TEDS

i) RAM

RAM stands for Random Access Memory. Information are stored in binary forms in RAM. Binary information is stored in memory in group of bits, each group of which is called word. A word is an entity of bits that moves in and out of memory as a unit- a group of 1's and 0's that represents a number, an instruction , one or more alphanumeric characters, or other binary-coded information. The capacity of memory unit is usually stated as the total number bytes that it can store. Communication between a memory and its environment is achieved through data input and output lines, address selection lines and control lines that specify the direction of transfer of information. A block diagram of memory is showed in Figure 4.0.

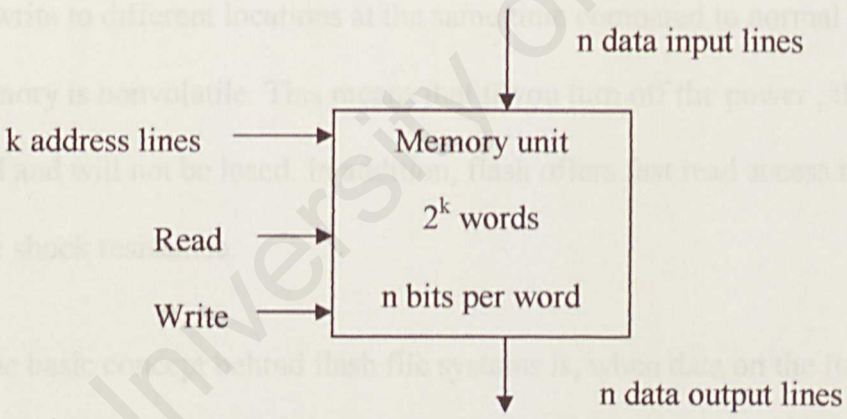

Figure 4.0 : Block diagram of memory

k address lines: specify the particular word chosen among the many available.

n data input lies: provide the information to be stored in memory.

n data output lines: supply the output lines coming out of memory.

Write: causes binary data to be transferred into memory.

Read: causes binary data to be transferred out of memory.

Memory unit: specified by the number of words it contains and the number of words in each word.

The TEDS are stored temporarily in RAM before it is transmitted to the Flash memory. RAM is volatile that is information is erased when the power is switched of f. The RAM serves as a buffer and receives data byte by byte and transmit it to the Flash.

ii) Flash memory

Flash memory is a form of EEPROM that allows multiple memory locations to be erased or written at one time. Normal EEPROM only allows one location at a time to be erased or written. This means that flash operates at higher speeds when the systems using it read and write to different locations at the same time compared to normal EEPROM. Flash memory is nonvolatile. This means that if you turn off the power , the informaton is retained and will not be losed. In addition, flash offers fast read access times and solid-state shock resistance.

The basic concept behind flash file systems is, when data on the flash store is to be updated, the file system will write a new copy of the changed data over to a fresh block, remap the file pointers, then erase the old block later when it has time.

Figure 4.1 shows the block diagram of ROM

Figure 4.1 : Block diagram of ROM

k inputs: provide the adress for the memory.

n outputs: give the data bits of the stored word that is selected by the address.

Figure 4.2 shows the relationship of TEDS, RAM and Flash memory

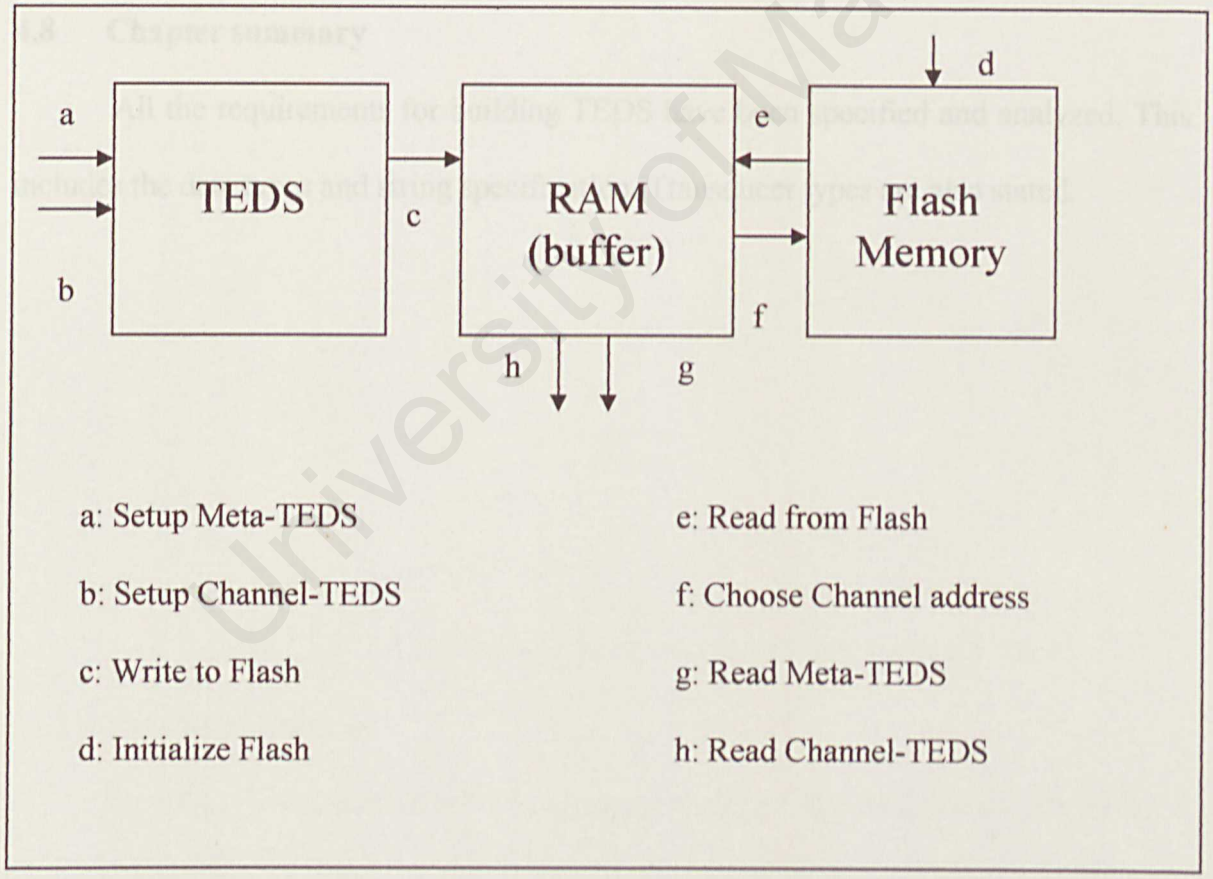

Figure 4.2 : Relationship of TEDS, RAM and Flash memory

For the data transport, the command sent will consist of 4 bytes and the data returned will consist of 6 bytes as shown below.

Command Sent	<Address>	<Control Data>	<Channel>	<Data>
Number of Byte	1	1	1	1

Data Returned	<Address>	<Data>	<Status>
Number of Byte	1	4	1

4.8 Chapter summary

All the requirements for building TEDS have been specified and analyzed. This includes the data types and string specification. Transducer types are also stated.

Chapter 5 : System Design

5.0 Chapter introduction

In general the system design portion of this project focuses on the development of TEDS mainly the Meta-TEDS and the Channel TEDS and how these data are kept in RAM and Flash memory.

5.1 TEDS

The 1451.2 are broken down into its logical software blocks as shown in Figure 5.0. For this project, the TEDS block will be done.

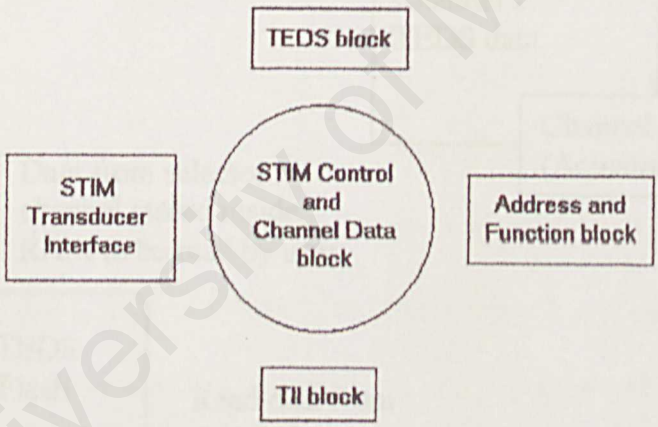

Figure 5.0 : 1451.2 broken down into it's logical software blocks

5.1.1 Data flow diagram

The Data Flow Diagram (DFD) represents the input of a data to a module or the output of data from a module. Figure 5.1 shows the data flow diagram for the whole TEDS.

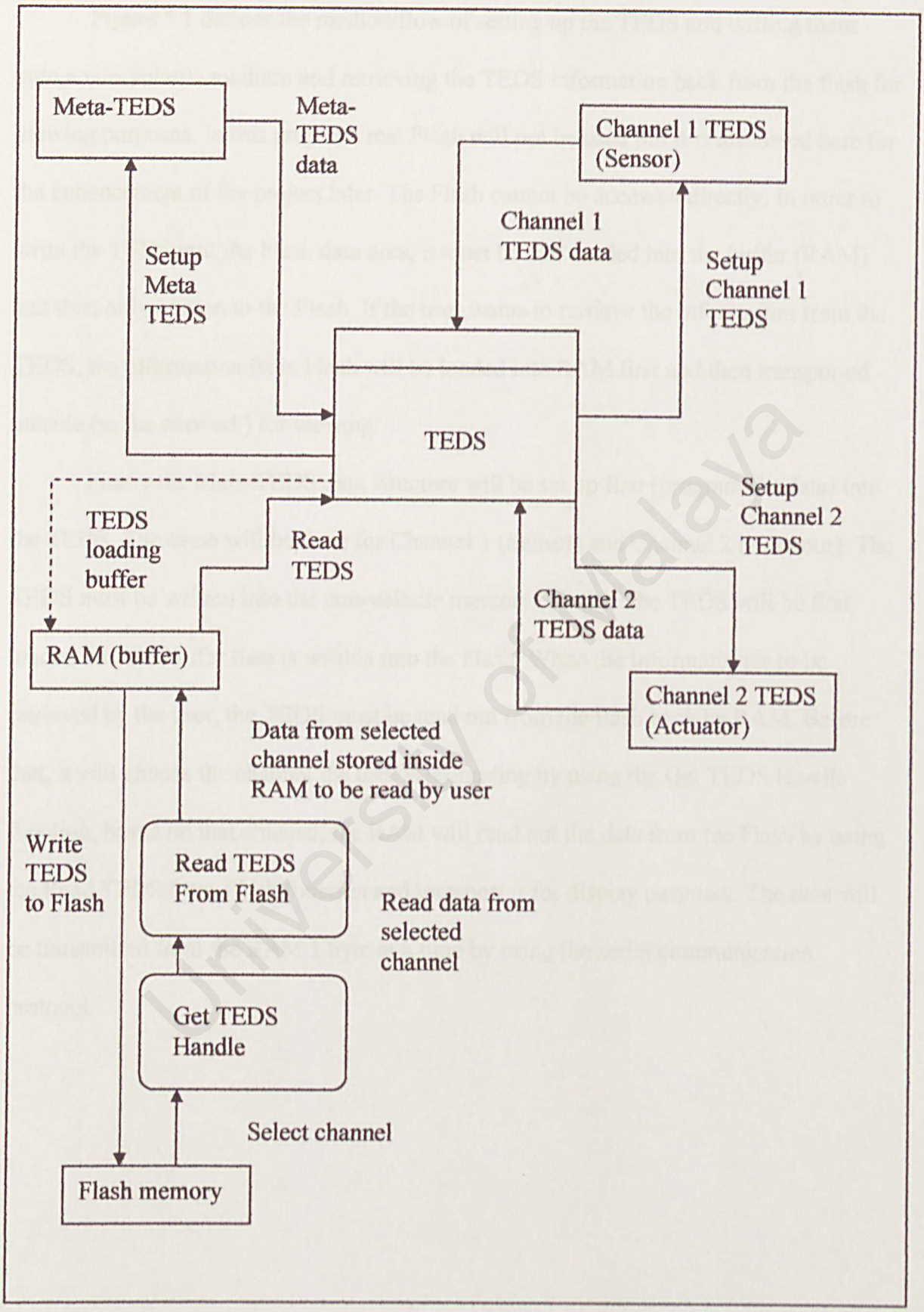

Figure 5.1 : DFD for TEDS

Figure 5.1 defines the method/flow of setting up the TEDS and writing them onto a non-volatile medium and retrieving the TEDS information back from the flash for viewing purposes. In this project a real Flash will not be used but it is discussed here for the enhancement of the project later. The Flash cannot be accessed directly. In order to write the TEDS into the Flash data area, it must be first loaded into the buffer (RAM) and then only written to the Flash. If the user wants to retrieve the information from the TEDS, the information from Flash will be loaded into RAM first and then transported outside (to the network) for viewing.

Firstly the Meta-TEDS data structure will be set up first (program the data) into the TEDS. The same will be done for Channel 1 (Sensor) and Channel 2 (Actuator). The TEDS must be written into the non-volatile memory (Flash). The TEDS will be first loaded into the buffer then is written into the Flash. When the information is to be retrieved by the user, the TEDS must be read out from the flash back by RAM. Before that, it will choose the channel the user is requesting by using the Get TEDS Handle function, based on that channel, the RAM will read out the data from the Flash by using the Read TEDS from Flash function and transport it for display purposes. The data will be transmitted from the RAM 1 byte at a time by using the serial communication protocol.

Figure 5.2 shows the block diagram for Meta-TEDS.

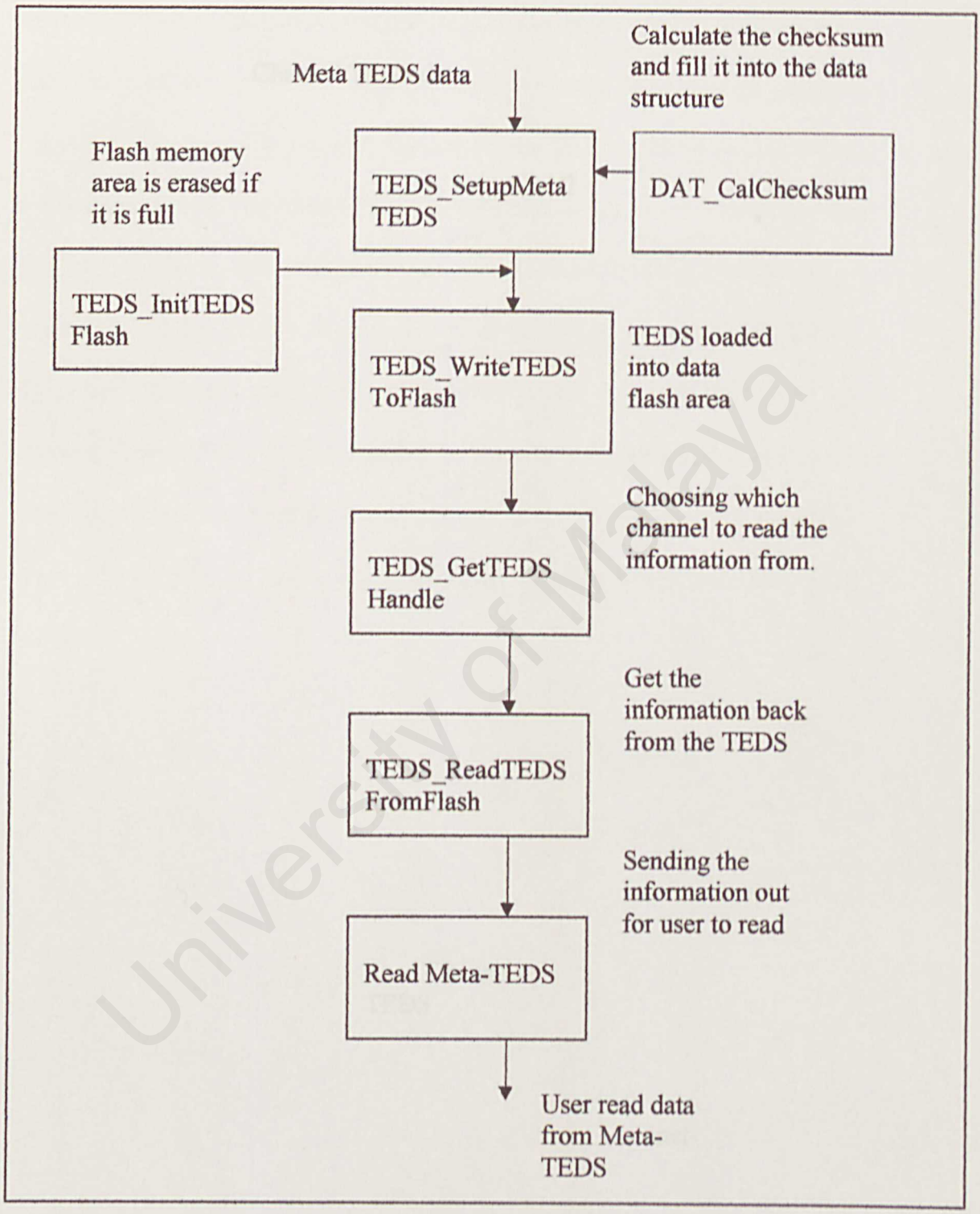

Figure 5.2 : Meta-TEDS Block Diagram

Figure 5.3 shows the block diagram for Channel-TEDS.

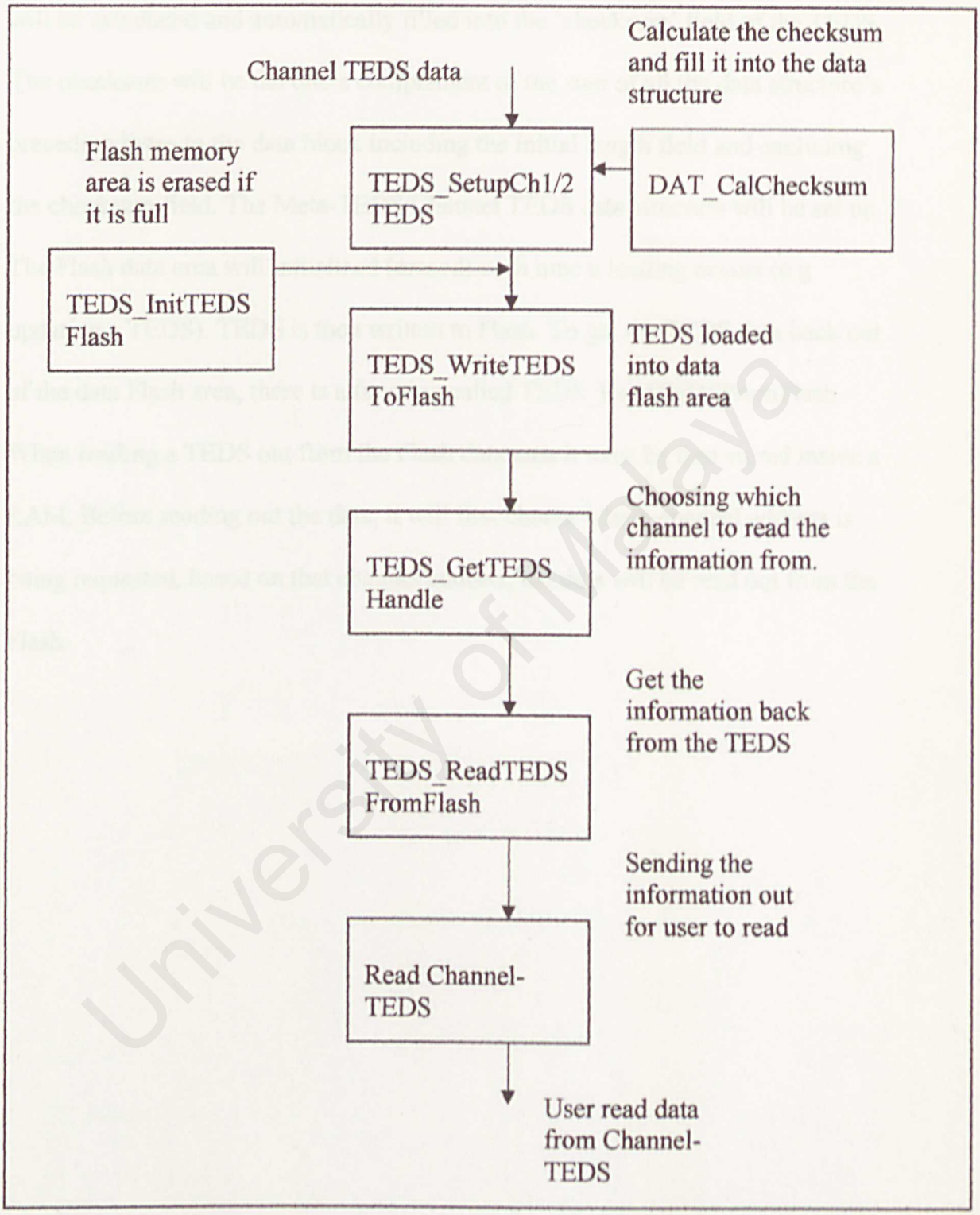

Figure 5.3 : Channel-TEDS Block Diagram

Based on Figure 5.2 and 5.3, firstly, the checksum for the set of data specified will be calculated and automatically filled into the 'checksum' field of the TEDS. The checksum will be the one's complement of the sum of all the data structure's preceding bytes in the data block including the initial length field and excluding the checksum field. The Meta-TEDS/Channel TEDS data structure will be set up. The Flash data area will initialized (erased) each time a loading occurs (e.g updating a TEDS). TEDS is then written to Flash. To get the TEDS data back out of the data Flash area, there is a function called TEDS_ReadTEDSFromFlash. When reading a TEDS out from the Flash data area it must be first stored inside a RAM. Before reading out the data, it will first choose which channel address is being requested, based on that channel address, the data will be read out from the Flash.

Figure 5.4 shows the flow chart of TEDS.

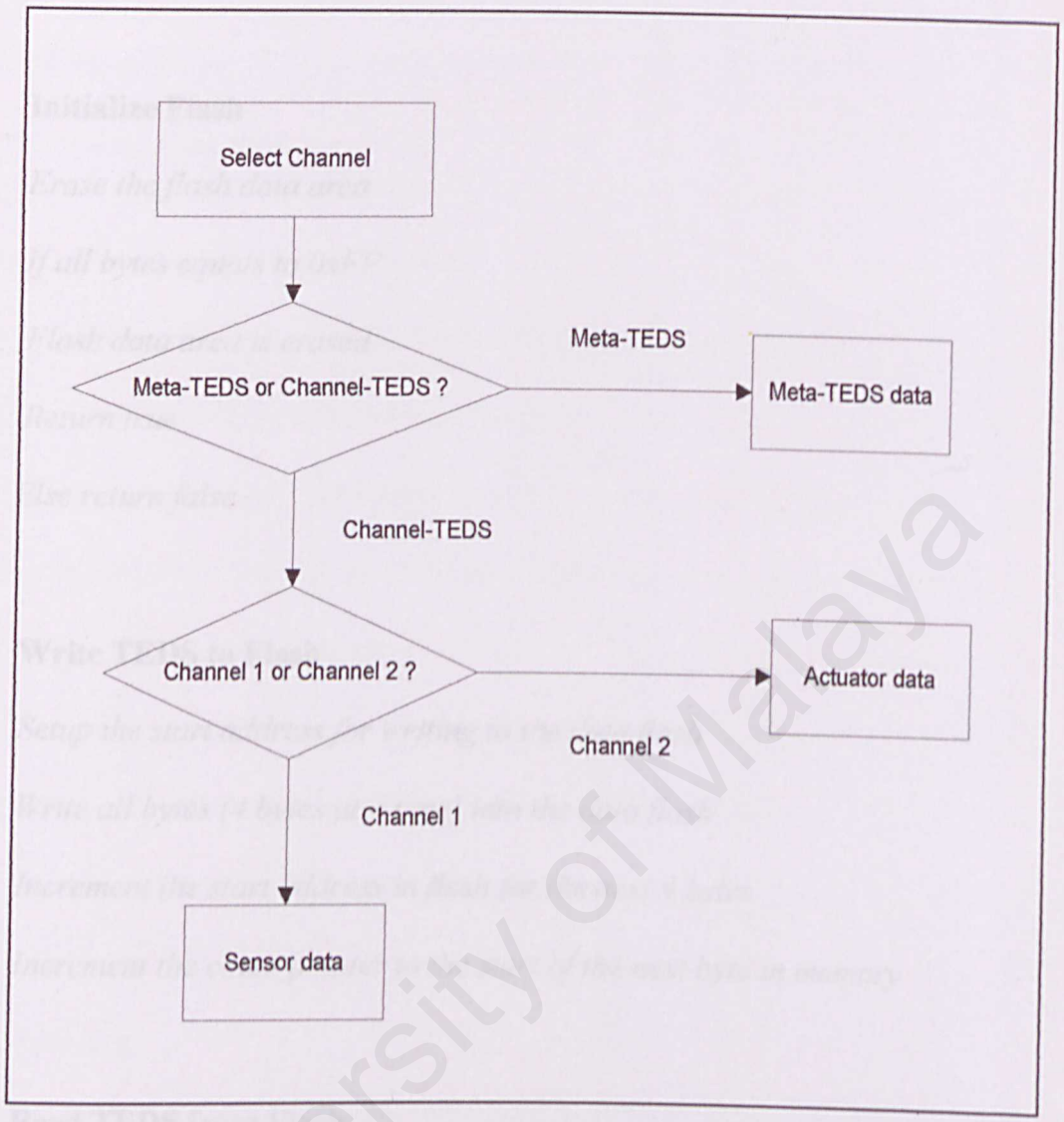

Figure 5.4 : Flow Chart of TEDS

5.1.2 Pseudocode

(i) Pseudocode for TEDS

Setup Meta-TEDS

Setup Meta-TEDS

Setup Channel 1 TEDS

Setup Channel 2 TEDS

Calculate checksum for all the channels

Initialize Flash

Erase the flash data area

If all bytes equals to 0xFF

Flash data area is erased

Return true

Else return false

Write TEDS to Flash

Setup the start address for writing to the data flash

Write all bytes (4 bytes at a time) into the data flash

Increment the start address in flash for the next 4 bytes

Increment the count pointer to the start of the next byte in memory

Read TEDS from Flash

Setup the start address for reading from the data flash

Read the first 4 bytes from the TEDS specified

Read all the bytes into the buffer (RAM)

Increment the start address in flash for reading of the next 4 bytes

Increment the count pointer to the start of the next byte in memory

Get TEDS Handle (Select Channel)

Select channel

Get data from flash

(ii) Pseudocode of the interface

Select TEDS data block (Meta-TEDS or Channel TEDS)

If data block equals to Meta-TEDS

Read Meta-TEDS data

If data block equals to Channel-TEDS

If Channel equals to 1

Read sensor data

else if channel equals to 2

Read actuator data

5.2 Memory

In this project only the Meta TEDS and the Channel TEDS will be implemented. That meant that one for each channel (Channel 1 TEDS and Channel 2 TEDS) and one that describes the entire system as a whole (the Meta TEDS) were required.

The Meta TEDS for this system is 76 bytes in total, and the Channel TEDS are 96 bytes each. They are mapped into the 640 bytes data flash area. The TEDS location in data flash is shown in Table 5.0.

TEDS	Data Flash Address	TEDS Length
Meta	0x00	76 bytes
Channel 1	0x13	96 bytes
Channel 2	0x2B	96 bytes

Table 5.0 : TEDS locations in data flash

(i) Data Flash

There are 640 bytes of data flash available:

- 76 bytes are required for the smallest version of Meta TEDS
- 96 bytes are requires for each of the smallest version of the channel TEDS
- up to 5 channels TEDS (plus Meta TEDS) can be stored in the data flash area

The TEDS will be arranged in the Flash Data Area. The flash data area is 640 (0x280) bytes long. It is accessed 4-bytes at a time (4-bytes pages). Each address references 4 bytes => the address space is 0x00 – 0xA0. TEDS are written starting from the first field, most significant byte first. See Figure 5.5.

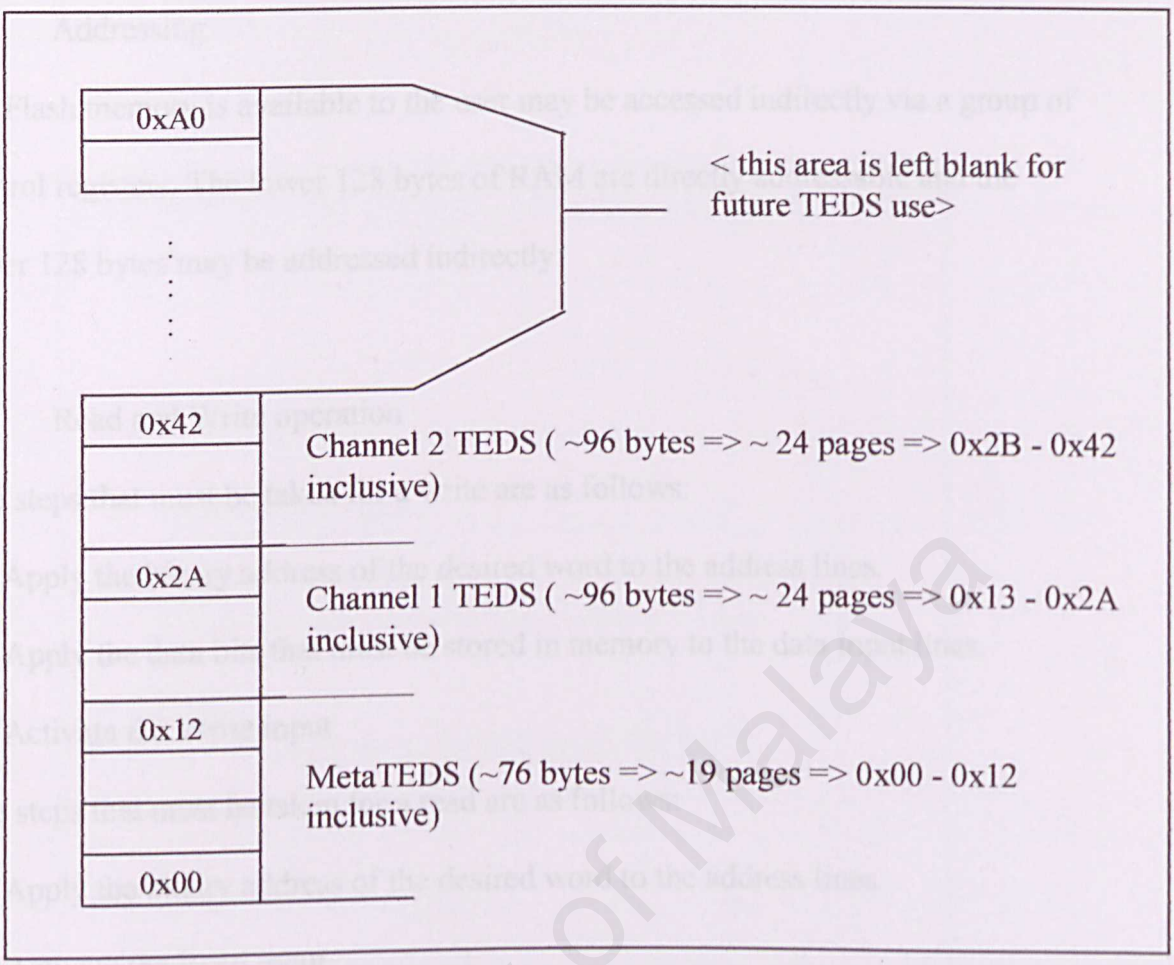

Figure 5.5 : The TEDS arranged in Flash Data Area

(ii) RAM (buffer)

If user request to read data, the data is read from flash and stored into RAM before it is sent to the user.

There are 256 bytes of RAM available:

- 100 bytes reserved for loading the TEDS
- 10 bytes reserved for each implemented channel
- the run- time data space requires over 30 bytes
- the run-time stack space requires between 20 and 30 bytes

iii) Addressing

The Flash memory is available to the user may be accessed indirectly via a group of control registers. The lower 128 bytes of RAM are directly addressable and the upper 128 bytes may be addressed indirectly.

iv) Read and Write operation

The steps that must be taken for a write are as follows:

1. Apply the binary address of the desired word to the address lines.
2. Apply the data bits that must be stored in memory to the data input lines.
3. Activate the Write input.

The steps that must be taken for a read are as follows:

1. Apply the binary address of the desired word to the address lines.
2. Activate the Read input.

5.3 Interface prototype

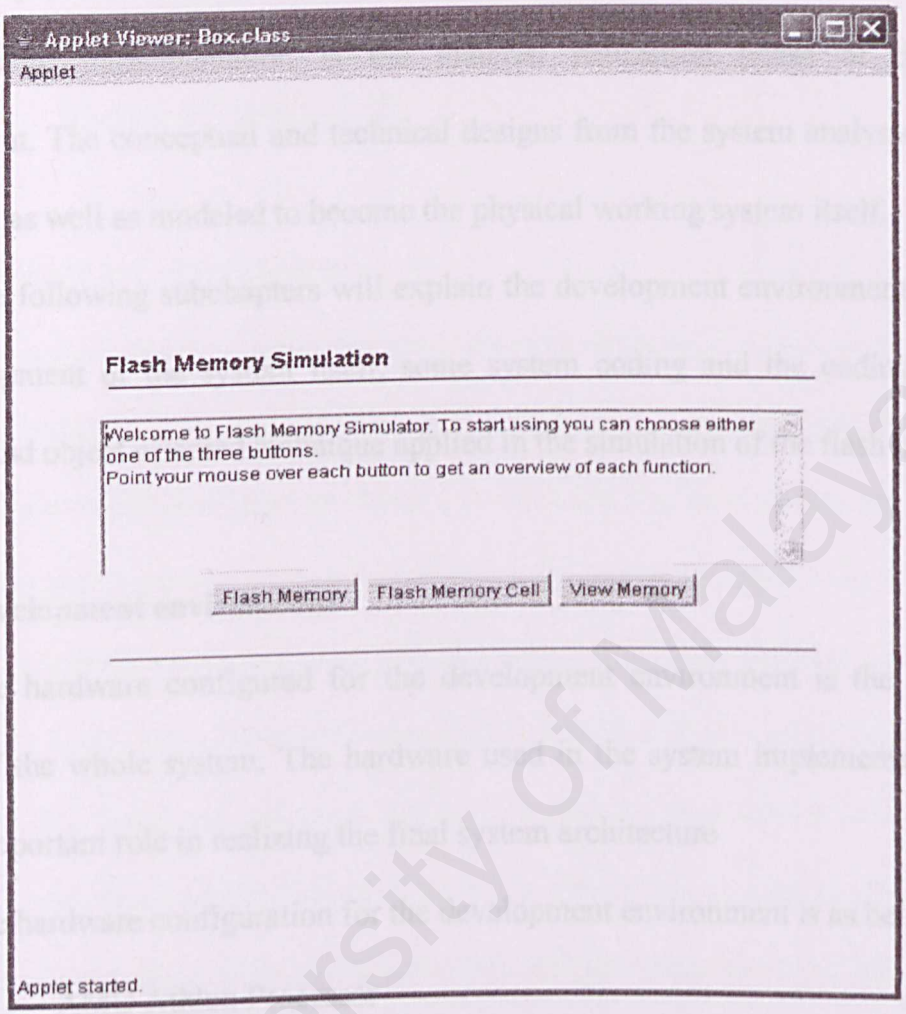

Figure 5.6 : Interface prototype

5.4 Chapter summary

All the design specification has been specified. This chapter also includes the interface prototype for the TEDS simulator.

Chapter 6: System Implementation

6.0 Chapter introduction

System implementation is the material realization phase of the system development. The conceptual and technical designs from the system analysis phase are interpreted as well as modeled to become the physical working system itself.

The following subchapters will explain the development environment as well as the development of the system itself, some system coding and the coding style and approach and object oriented technique applied in the simulation of the flash memory.

6.1 Development environment

The hardware configured for the development environment is the underlying element of the whole system. The hardware used in the system implementation phase plays an important role in realizing the final system architecture

The hardware configuration for the development environment is as below:

- AMD Athlon Processor
- 654 Mhz
- 128 MB RAM

6.1.1 Software development environment

Hardware and software form a tightly coupled cohesion that operates in unison to performed programmed tasks. Without software, the fastest, biggest or the most powerful computer will also be inoperative and idle in the corner.

The software tools utilized in the development environment are listed as below:

- Java 2 SDK Software Development environment version 1.4.2_06
- JCreator LE version 2.50

6.2 Development of the system

To be able to use Java as the programming language to code or to develop a program or a system, one needs to gain an understanding of the concepts of the objects oriented. Understanding of what an object is, what a class is, how objects and classes are related, how objects communicate by using messages is much needed.

In definition an object is a software bundle related variables and methods. "Objects" is a key to object oriented technology. One can look around now and see many examples of real-world objects: a car, a bicycle, a desk a, television set and a computer. These real world objects share two characteristics: they all have a state and behavior. For example cars have a state (brand, color, size) and behavior (accelerating, braking).

Software objects are modeled after real world objects in that they too have state and behavior. A software objects maintain its state in one or more variables. A variable is an item of data named by an identifier. A software objects implements its behavior with methods. A method is a function associated with an object.

Methods surround and hide the objects nucleus from other objects in the program. Packaging an object's variables within the protective custody of its method is called encapsulation. Encapsulating related variables and methods into a neat software bundle is a simple yet powerful idea that provides two primary benefits to software developers:

- Modularity: the source code for an object can be written and maintained independently of the source code for other objects. Also, an object can be easily passed around in the system. One can give a bicycle to someone else and it still work.
- Information hiding: an object has a public interface that other objects can use to communicate with it. The object can maintain private information and methods that can be changed at any time without affecting the other objects that depend on it. One does not need to understand the gear mechanism in his bike to use it.

A single object is generally not very useful. Instead, an object usually appears as a component of a larger program or an application that contain many other objects. Software objects interact and communicate with each other by sending messages to each other.

A class is a blueprint or prototype that defines the variables and the methods common to all objects of a certain kind. For example, a bicycle is just one of the many bicycles in the world. In object oriented software it's also possible to have many objects of the same kind that share characteristics: rectangles, employee records, video clips and so on. A software blueprint for objects is called a class.

6.2.1 System coding

After researches and studies have been done, a decision was made to code the simulation system using the Java programming language, and to be able to run the simulation as a standalone windows application and as an applet which can be executed

using the Xinox Software's JCreator Light Edition (LE) v2.50 integrated development environment.

The flash memory simulation system utilizes three java platform packages. The three packages are java.awt , java.applet, and java.swing.

Package java.awt and java.swing. contains all of the classes for creating user interfaces and for painting graphics and images. A user interface object such as a button pr a scrollbar is called, in AWT terminology, a component. The Component class is the root of all AWT components. Some components fire events when a user interacts with the components. The AWTEvent class and its subclasses are used to represent the events that AWT components can fire. The java.swing. is latter version.

A container is a component that can contain components and other containers. A container can also have a layout manager that controls the visual placement of components in the container. The AWT package contains several layout manager classes and an interface for building your own layout manager.

Package java.applet provides the classes necessary to create an applet and the classes, an applet uses to communicate with its applet context.

The applet framework involves two entities: the applet and the applet context. An apple is an embeddable window with a few extra methods that the applet context can use to initialize, start, and stop the applet.

The applet context is an application that is responsible for loading and running applets. For example, the applet context could be a web browser or an applet development environment.

After deciding of the packages and Java-built in classes that can be used for the simulation system, it was time to decide on the entities or the classes that can represents the simulation of the flash memory.

The coding f this Java simulation system after its final enhancement and refinement, is divided into 4 main modules:

- Main module
- Animate module which consists of read and write module
- Cell module which consists of read cell and write cell module
- Memory structure module without animation which consists of edit class and fill class.

The main module will call all the other modules. The animate module which consists of read and write module consists of the memory schematic. This memory schematic will receive user input of unsigned data and performs the memory functions of reading data, writing data and erasing data.

The cell module in the other hand performs 1 bit of data. That is in the read module, it will show how a bit of data is received and moved along the write and bit line and also the when the transistor is turned on in order to display the data. The data will be displayed to be read after the transistor is turned on. The write cell will perform the opposite. The data will only be stored inside the memory after the transistor is turned on.

The memory structure module will perform the same function as in the animation module. The only difference here is that it shows the structure of the memory and in what shape the data will be kept and there will be no animation.

6.3 Coding style

6.3.1 Formatting and indenting codes

Formatting and indenting codes are constantly associated to good coding practice. A code that is written without proper formatting or indenting will function or what as well as a formatted code. However, this can make exceptionally difficult to see where an error is coming from. Indentation principally makes the structure of the code stand out and easier to be read. This eventually will help in detecting and removing the common programming errors.

Creator provides user a good formatting and indenting facility where user is associated to format and indent codes automatically in the environment while coding a Java source.

6.3.2 Commenting codes

Comments are part of this program code. It is a good and recommended practice. Commenting will help the reader of the code to understand what and why the coding was written .In addition, this also makes it easier for others especially collaborating programmers to understand the coding.

Chapter 7: System Testing

7.0 Chapter introduction

System testing is a crucial phase in the development of the simulation of the flash memory, as it tells whether the coding of the system is successfully implemented, whether the executing system visualizes the flash memory accurately and whether the code needed to be modified, enhanced, deleted, added or debugged. Testing is done throughout the system development and not just at the end.

7.1 Compiling and executing

Once the coding of the system is completed, and all the classes designed are fully coded, the java source needs to be compiled to see whether there any bugs or errors in the coding.

If the java application can be successfully compiled without any error, then the testing phase can proceed to executing the application.

Otherwise, if the java application is compiled with errors, the testing phase needs to jump to the debugging phase, before it is recompiled again.

7.2 Debugging

Once the application is compiled with errors, the error message need to be scrutinized to identify where the errors have occurred on the source code. The error might be caused by syntax mistakes, such as left out of semi-colon, curly braces and any other symbols. Some errors might also be caused by logical errors such as errors in dereferencing, errors in calling methods, or errors in passing arguments.

The process of debugging is to check on those mistakes and correct them. It is a process eliminating errors or bugs from the source code in order for the system to compile successfully.

7.3 Accuracy of execution

After the source code has all the errors eliminated and compiled successfully it will then be executed or in other words run. The target of he execution is for the users to use the system or interact with the system through the system interface.

In the context of flash memory simulation, the compiled source is executed checked and verified if the simulation acts according to the use'sr input or not. There may be logical errors where source code compiled successfully but simulation does perform according to the users input. My tester was my supervisor who checked the system to verify the correctness of my system with valid data.

If the system does not act properly according to the user's response, the testing phase will go back to debugging-compiling-executing process until the system is able to work accurately according to user's input.

7.4 Unit testing

Unit testing verifies that the component functions properly with the types of input expected from studying the component's design. The first step is to examine the program code by reading through it, trying to spot data and syntax faults. This is followed by comparing the code with specifications and with the design to make sure that all the relevant cases have been considered. Next view the result and eliminate remaining syntax faults if necessary. Finally test cases are developed to show that the

input is properly converted to the desired output. Unit testing tries to look for all the possible errors that will occur in a program. A complete test process should test all of the following categories of test data.

- Normal data - to test a given correct data will produce the expected results.
- Erroneous data - for a given erroneous data, like invalid date format, does the system detect or not.
- Boundaries value analysis - data that are out of the range specified will be used to test the system because errors may occur at the extreme point.

7.5 Module testing

When individual components are working properly and meets the objectives, these components are combined into a module. A module is a collection of dependent components. Module testing enables each module to be tested independently. This testing will ensure that the module calling sequences in this project is systematic

7.6 Integration testing

When the individual components are working correctly and meet the objectives, these components are integrated into a working system. In other words, integration testing is the process of verifying that the system components work together as described in the system and program design specification.

Integration testing is used on flash memory simulation for constructing the program structure while at the same time conducting tests to uncover errors associated

with interfacing. The objective is to take unit-tested modules and build a program structure that has been dictated by design. This testing will ensure that the interfaces in the simulation are systematized and linked to the correct document of the system.

7.7 System testing

The last testing procedure done is the system testing. Testing the system is very different from unit testing and integration testing. The objective of unit testing and integration testing is to ensure that the code has implemented the design properly. In other words, the code is written to do what the design specifications intended. In the system testing, a very different objective is to be achieved, that is to ensure that the system fulfills user requirements or my supervisor's requirements.

Testing performed:

- Function testing

Function testing is based on the system's functional requirements. The testing is carried out for the system's every module. Each module is tested individually to determine whether the system performs as required.

- Performance testing

Performance testing addresses the non-functional requirements of the application. The types of performance tests carried out for this application are:

- Volume tests

  The data field and address field are checked to see if they can accommodate all expected data.

- Timing tests

  System performance is timed to ensure that it meets user's requirements.

7.8 Chapter summary

During the testing phase, several testing strategies were being used to ensure the system is integrated and developed successfully. Approaches were employed to recover faults in the system. Unit, module, integration and system testing has been carried out for this system. The objective of a system will only be achieved after all the thorough testing done by the different user with different aspects.

Chapter 8: System Evaluation and Conclusion

8.0 Chapter introduction

This chapter explains all the evaluation procedures taken to identify simulation's strengths and limitations. Throughout the flash memory simulation development process, a lot of problems were encountered and solved eventually. This chapter also discusses future enhancement to the system.

8.1 System evaluation

This section discusses the expectations achieved from the initial goals set at the beginning of this project. The systems strengths and limitations are also listed. The main problems encountered whilst developing this simulation is also explained along with all the knowledge gained since the commencement of this project.

8.1.1 Expectations achieved

In general, the initial expectations set by the set by the project objectives had been achieved. The following are the achieved goals:

i. All the basic function performed by flash memory like read, write, and clear.

ii. Unsigned integer to be entered as the data.

iii. Flash memory cell simulation.

8.1.2 System limitations

Although the system performs all the basic function, there are limitations in the system. The limitations are as follows:

i. The system only takes unsigned data, it does not take floating point and signed data.

ii. The system only reads the hardcoded data in the program. It would not read a newly entered data.

iii. For the flash memory cell, there are only cell for read and write zero. There is no read and write one.

8.1.3 Problems encountered

During the implementation of the flash memory simulation, a few problems were encountered.

i. Problem with the programming language.

I had no prior knowledge in the java programming language before the implementation of the system started. I needed to get a deep detailed idea of object oriented programming in java before coming out with the architectural basis design of the flash memory simulation. Objects of the system were to be identified and verified of their applicability and flexibility to be coded. Once the coding of the system started, I encountered a lot of syntax and logical errors, which consequently, forced me to spend a huge amount of time in debugging and even seeking help from experts in forums.

There are a lot of built in classes in Java. It was a bit confusing to use which class and I took some time to learn about all the classes. Flash memory simulation involves using animation and learning how to do animation in java

took most of my time. There are no drag and drop function in Java. So I had to draw all the schematics by programming which was difficult to do.

ii. Insufficient detailed references

There were many references on java but none really taught me how to program a memory. So I did the system using only examples (without codings) and based on my own understanding. Therefore the end result was not as good as expected.

iii. Integration problem

The system was done part by part or module by module. All modules then were integrated. During the integration, I faced a lot of problems. Integrating two of the most important modules that is the read and write module was the most difficult part. The system could not run after I integrated that two modules. I did not know the correct techniques for the integration. After much research, examples and help from my friends then only I could integrate all the modules.

8.1.4 Knowledge gained

During the whole project I learned a lot of things. Firstly of course I learned the Java language itself. Before the project I did not know a thing about Java. This project made me learn a new programming language which will be an advantage for me. Not only that, I also learned how to do animation using java.

Secondly, I had a better understanding of how the memory works and how it works in terms of binary digits that is how it changes all the data including characters and numbers to binaries. I also learned how the flash cell works.

8.2 Project enhancement

The development of any system is always a dynamic and ongoing process. Furthermore, new ideas and improvements constantly keep on coming whilst the system is being developed. However, due to constraints and problems encountered, some of the planned implementations could not be developed for the system.

The following future enhancement can be made to the system:

a) Make the system to receive signed and floating point numbers.

b) The system must also be able to receive characters instead of numbers only.

c) The flash cell must be made for read and write 1.

d) Currently the system only shows the symbol of the flash cell, hopefully the flash circuitry can be made in the future.

8.3 Discussion

During the two semesters of this thesis a lot of things have been changed. Things that have been changed are as follows:

1) The system that has been done is totally concentrated on Flash memory and nothing concerning TEDS. This includes on how data is stored or read to and from Flash memory because the TEDS is kept inside the Flash memory. TEDS were not concentrated because a lot of time was taken to study the Flash behavior and to learn how to make the animation in Java.

2) Most of the data in TEDS were in μ units but the system was done only for unsigned decimal data. This was because converting μ units to binary was difficult to do and so I decided to concentrate on decimal data first.

References

[1] IEEE Std 1451.2-1997, "IEEE Standard for a Smart Transducer Interface for Sensors and Actuators – Transducer to Microprocessor Communication Protocols and Transducer Electronic Data Sheet (TEDS) Formats", IEEE Instrumentation and Measurement Society, TC-9 Committee on Sensor Technology, Institute of Electrical and Electronics Engineers, New York, N.Y., Sept. 1998.

[2] IEEE-P1451.2 Smart Transducer Interface Module, *http://ieee1451.nist.gov/senoct11-2col.pdf*

[3] A Smart Transducer Interface for Sensors and Actuators, *http://ieee1451.nist.gov*

[4] Sensor Industry Developments and Trends, *http://www.sensorsmag.com/sensors-cgi*

[5] Reading and Writing TEDS with the LabVIEW PDA Module, *http://zone.ni.com/zone/jsp/zone.jsp*

[6] Reaching Up and Out : Interface Standards Smarten Up Sensors, Transducer, *www.controleng.com/*

[7] IEEE 1451 Family Of Smart Transducer Interface Standards, *www.kistler.com*

[8] IEEE 1451 Smart Transducer Interface, *www.analog.com/Analog_Root/sitePage/mainSectionHome*

[9] IEEE-P1451 Draft Standard for Smart Transducer Interface for Sensors and Actuators,
http://ieee1451.nist.gov/group1.htm

[10] IEEE 1451 Overview "A Smart Transducer Interface for Sensors and Actuators",
http://ieng9.ucsd.edu/~groff/cse237a/IEEE1451.html

[11] Article : The Importance of IEEE 1451T,
http://ethernet.industrial-networking.com/articles/i01_ieee1451rvness.asp

[12] Building IEEE 1451.2 Smart Transducer Interface Modules (STIMs),
http://www.telemonitor.com

[13] An Overview of the IEEE 1451.4 Transducer Electronic Data Sheet (TEDS),
http://zone.ni.com/devzone/conceptd.nsf/webmain/2B71D966B0E41FB586256D9B0068BCBC?opendocument

[14] Network strategies make sense,
http://www.isa.org/InTechTemplate.cfm?Section=Article_Index&template=/ContentManagement/ContentDisplay.cfm&ContentID=27178

[15] Smart Transducers - Principles, Communications, and Configuration,
http://www.vmars.tuwien.ac.at

[16] Sensor STIM,
http://ieee1451.nist.gov/stimsen.html

[17] Sensor/Actuator STIM,
http://ieee1451.nist.gov/stimsensact.html

[18] TEDS - plug and play sensor configuration,
http://www.sensotec.com/pnpterms.shtml

[19] SIM Object Model,

http://www.isd.mel.nist.gov/documents/rippey/ngisAPI/i,iireqspec.html

[20] IEEE p1451.2 Smart Sensor Interfaces,

http://eesensors.com/tutorial.html

[21] The Smart Transducer Interface Standard (IEEE P1451),

http://grouper.ieee.org/groups/1451/2

[22] IEEE 1451: Smart Sensor Networking,

http://www.andrew.cmu.edu

[23] ISO 8859-1,

http://en.wikipedia.org/wiki/ISO_8859-1

[24] Sensors smarten up,

http://www.reed-

electronics.com/ednmag/article/CA56680?pubdate=3%2F4%2F1999

[25] Flash memory,

http://en.wikipedia.org/wiki/Flash_memory

[26] Simulation of MOS Memory Operation,

http://jas2.eng.buffalo.edu/applets/education/system/memory/index.html

---

The references and links to website have been checked correct as of 25th February 2005.

Glossary

A

acknowledgment

A signal that is used to reply to a message or signal originator that its message or signal was received.

actuator

A transducer that accepts an electrical signal and converts it into a physical action.

address

A character or group of characters that identifies a register, a particular part of storage, or some other data source or destination. This standard uses *functional addresses* and *channel addresses* to control the flow of data and configuration information between the Network Capable Application Processor and the Smart Transducer Interface Module.

analog to digital converter

A circuit whose input is information in analog form and whose output is the same information in digital form.

B

buffer

An intermediate data storage location used to compensate for the difference in rate of flow of data or time of occurrence of events when transmitting information from one device to another.

buffered channel

A channel in which the data is placed into a buffer prior to a trigger event and then

transmitted or acted upon following that trigger event. This contrasts with an unbuffered channel in which the data is not taken by, or available to, the channel until following the trigger event.

byte

Eight bits of data. Synonym: octet

C

calibration

The determination of the data to reside in the Calibration Transducer Electronic Data Sheet and to be used for correction.

channel

A single flow path for digital data or an analog signal, usually in distinction from other parallel paths. An IEEE 1451.2 channel provides a path for a single commodity or logical state, either real or virtual, using a single data model and a single set of physical units.

channel address

The portion of a full data transport address that specifies the channel to which the read or write is directed.

channel groupings

Channel groups are manufacturer specifications that define the inherent relationships between the channels of a multichannel Smart Transducer Interface Module. This grouping information is not normally used by the Smart Transducer Interface Module itself. This information will normally be used by Network Capable Application Processor applications to properly compose human readable displays or in formulating other computations. For example, channel groupings can be used to indicate which channels represent the three vector axes of a three-axis vector measurement.

correction

The evaluation of a multinomial function using information from the Calibration Transducer Electronic Data Sheet together with data from one or more channels

D

data conversion

The translation of data from one numeric form into another (e.g., converting a digital-to-analog converter input bit stream into a voltage).

data model

The numeric format in which the Smart Transducer Interface Module will output or accept data.

data sequence sensor

A sensor that samples data independent of any triggers from a network capable application processor.

data sheet

A set of information on a device that defines the parameters of operation and conditions of usage (usually produced by the device's manufacturer).

data structure

A group of digital data fields organized in some logical order for some speciÞc purpose. A two-dimensional paper version of a data structure is an empty Þll-in-the-blanks form or an empty tabular chart with organized column and row headings. A data structure is the template by which data is stored in computer memory.

digital interface

A set of wires and a protocol for transferring information by binary means only.

digital to analog converter

A circuit that converts an input number sequence (digital) into a function of a continuous variable (analog).

E

electronic data sheet

A data sheet stored in some form of electronically readable memory (as opposed to a piece of paper).

enumeration

The listing of the meaning associated with each binary numeric value possible in a data field's storage. Binary numbers are usually expressed in decimal terms for human convenience. Not all possible numeric values need have a specific meaning. Values without meaning are declared to be unused or reserved for future use. Enumeration is the process of declaring the encoding of human interpretable information in a manner convenient for digital electronic machine storage and interchange. Each transducer electronic data sheet data field that is of data type *enumeration* shall contain a table that defines the meaning of the data field for each binary number possible. The meanings encoded in each data field shall be specific and unique to that data field and only that data field. The value becomes meaningless if not associated with the data field and its defining table.

event sequence sensor

A sensor that detects a change of state in the physical world. The instant in time of the change of state, not the state value, is the "measurement".

F

full data transport address

The combination of a *functional address* and a *channel address*, that specifies whether data is being read or written, to which function, and to which channel.

functional address

The portion of a full data transport address that specifies the read or write function that is to be performed.

H

hot swap

The act of connecting or disconnecting a Smart Transducer Interface Module and a Network Capable Application Processor without first turning off the power that the Network Capable Application Processor supplies to the Smart Transducer Interface Module over the Transducer Independent Interface.

I

interrupt

A signal for a processor to suspend one process and begin another. As implemented in IEEE Std 1451.2-1997, an interrupt is a signal from the Smart Transducer Interface Module that enables it to request service from the Network Capable Application Processor.

L

least significant bit

The bit in the binary notation of a number that is the coefficient of the lowest exponent possible.

M

meta-

A Greek prefix meaning that which pertains to the whole or overall entity or that which is in common or shared with all member entities comprising the whole.

Meta-TEDS

The collection of those Transducer Electronic Data Sheet data fields that pertain to the whole or overall entity or those that are in common or shared with all member entities (channels) comprising the whole transducer product.

most significant bit

The bit in the binary notation of a number that is the coefficient of the highest exponent possible.

multinomial

A linear sum of terms involving powers of more than one variable.

N

negative logic

An electronic logic system where the voltage representing one, active, or true has a more negative value than the voltage representing zero, inactive, or false. Also known as negative-true logic, it is normally used in electronic and computing data and communications switching systems for noise immunity reasons.

Network Capable Application Processor

A device between the Smart Transducer Interface Module and the network that performs network communications, Smart Transducer Interface Module communications, and data conversion functions.

not a number

As defined in IEEE Std 754-1985, a bit pattern of a single or double precision real number data type that is a result of an invalid floating point operation.

O

octet

A group of 8 bits, also known as a byte.

P

pacing

A method to regulate the ßow of bytes read from or written to a Smart Transducer Interface Module.

positive logic

An electronic logic system where the voltage representing one, active, or true has a more positive value than the voltage representing zero, inactive, or false. It is normally used in industrial and commercial control switching systems for safety reasons.

R

read frame

The transfer of data from a Smart Transducer Interface Module to a Network Capable Application Processor.

S

sensor

A transducer that converts a physical, biological, or chemical parameter into an electrical signal.

setup time

The period of time during which a system or component is being prepared for a specific operation.

signal conditioning

Sensor signal processing involving operations such as amplification, compensation, filtering, and normalization.

smart actuator

An actuator version of a smart transducer.

smart sensor

A sensor version of a smart transducer.

smart transducer

A transducer that provides functions beyond those necessary for generating a correct representation of a sensed or controlled quantity. This functionality typically simplifies the integration of the transducer into applications in a networked environment.

Smart Transducer Interface Module

A module that contains the Transducer Electronic Data Sheet,
logic to implement the transducer interface, the transducer(s) and any signal conversion or signal conditioning. This standard expressly requires that no operating mode of the Smart Transducer Interface Module ever permit these components to be physically separated. They may, however, be separated during manufacturing and repair.

T

transducer

A device converting energy from one domain into another. The device may either be a sensor or an actuator.

Transducer Electronic Data Sheet

A data sheet describing a transducer stored in some form of electronically readable memory.

Transducer Independent Interface

The digital interface used to connect a Smart Transducer Interface Module to a Network Capable Application Processor.

transducer interface

The physical connection by which a transducer communicates with the control or data systems that it is a member of, including the physical connector, the signal wires used and the rules by which information is passed across the connection.

transfer

To transmit, or copy, information from one device to another.

trigger

A signal to start an action. As defined in IEEE Std 1451.2-1997 a trigger is a signal from the Network Capable Application Processor serving as a command to the Smart Transducer Interface Module for an action to occur.

trigger cycle

A complete cycle comprising the assertion of the trigger signal by the Network Capable Application Processor followed by the acknowledgment by the Smart Transducer Interface Module.

V

virtual channel

A channel that behaves as a transducer from the point of view of the Network Capable

Application Processor even though nothing outside of the Smart Transducer Interface Module is sensed or changed. Virtual channels are useful for setting or reading operating parameters of other channels.

W

write frame

The transfer of data from a Network Capable Application Processor to a Smart Transducer Interface Module.

Appendix A : User Manual

Table of Contents

List of Figures

A. Introduction

Flash memory simulation system is a simulation of how the flash memory works in terms of its memory schematic and the memory cell.

B. Hardware specifications

Below are the minimum hardware requirements to run the Flash memory simulation:

- Windows XP Professional.
- AMD Athlon Processor.
- 654 Mhz.
- 128 MB RAM.

C. Software specifications

Below are the softwares required to run the Flash memory simulation:

- Java 2 SDK Software Development environment version 1.4.2_06 (J2SDK 1.4.2_06).
- JCreator LE version 2.50.

D. I) Installing J2SDK 1.4.2_06

- Download the J2SDK from the http://www.java.sun.com website. It is a free software so it does not need to be licensed. Save it in your hardrive.
- Double click the on the **Setup** launcher. The **InstallShield Wizard** will appear as in Figure A1. Please wait for the **license** screen to appear as in Figure A2.

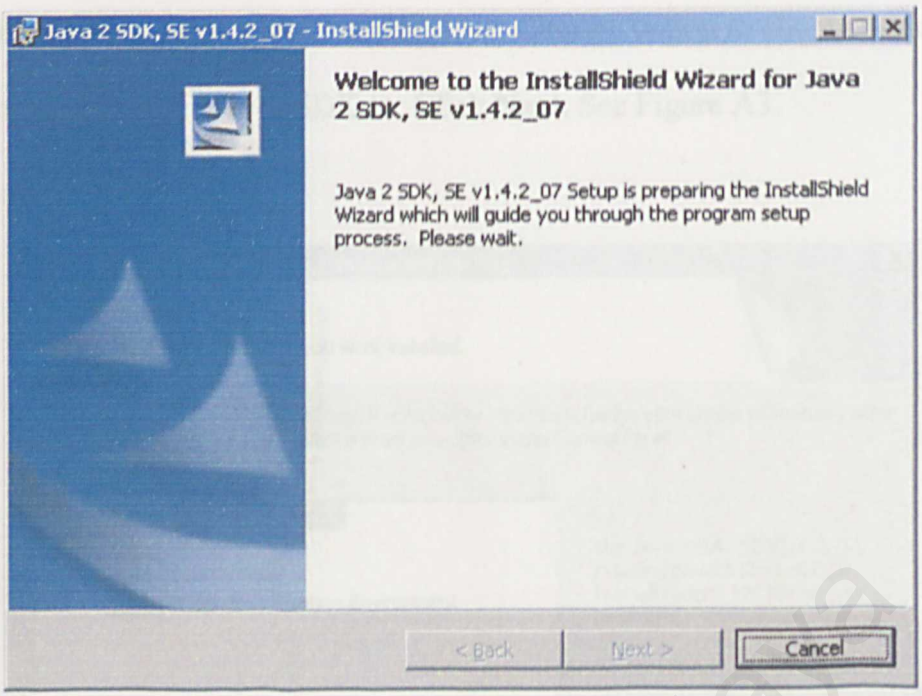

Figure A1

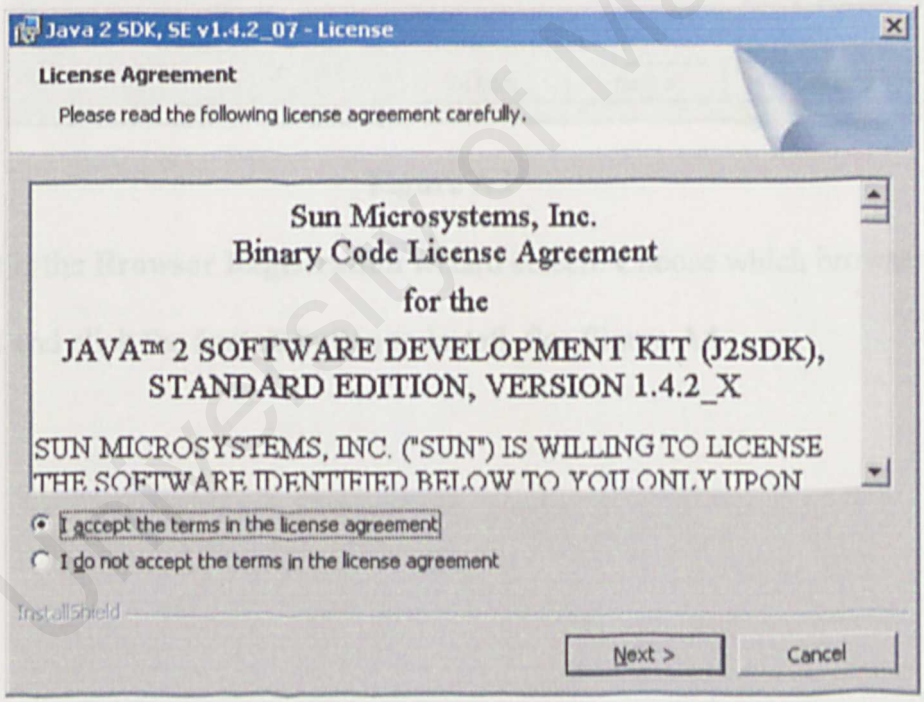

Figure A2

- Click the **I accept the terms in the license agreement** radio button and click **Next** button.

- In the **Custom Setup** wizard screen, click **Change** button to choose which directory to install your J2SDK to. Click **Next.** See Figure A3.

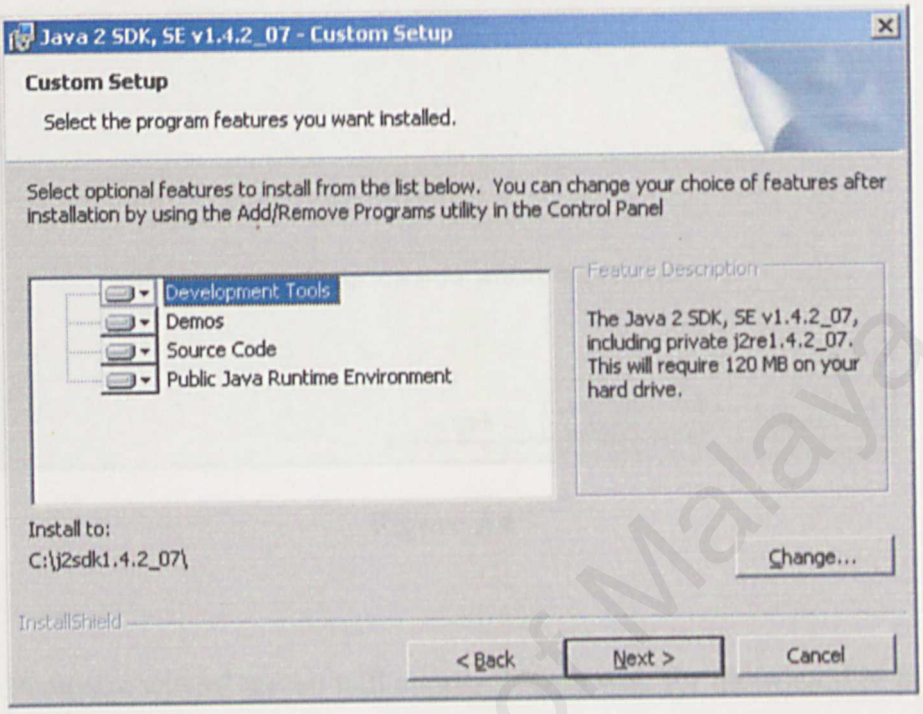

Figure A3

- Next is the **Browser Registration** wizard screen. Choose which browser you want and click the **Install** button to install. See Figure A4.

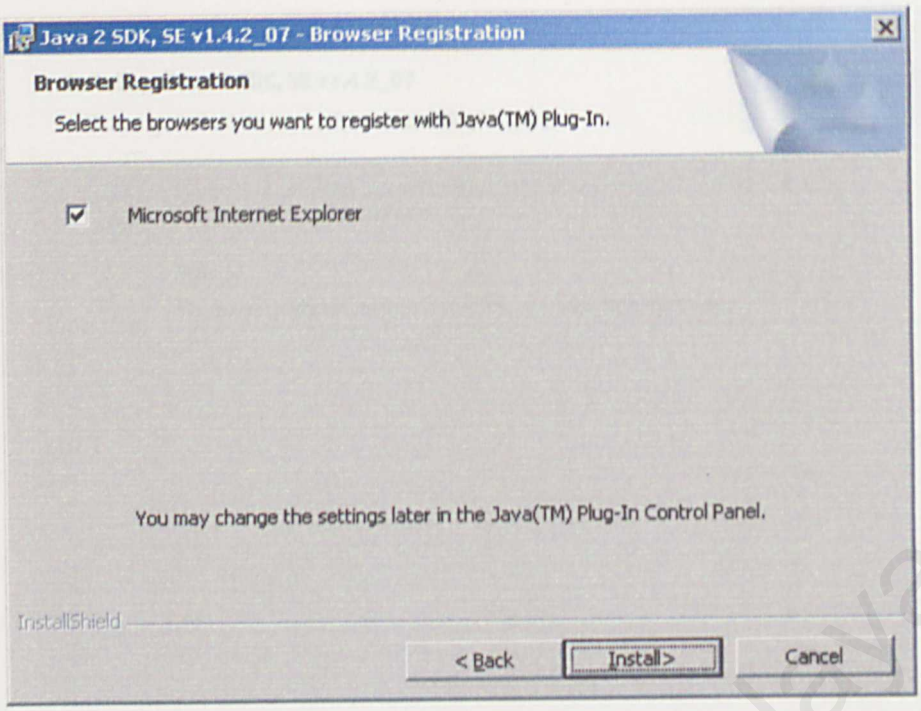

Figure A4

- The **Progress** wizard screen will appear. Please wait for the wizard to install Java in your computer. See Figure A5 and Figure A6.

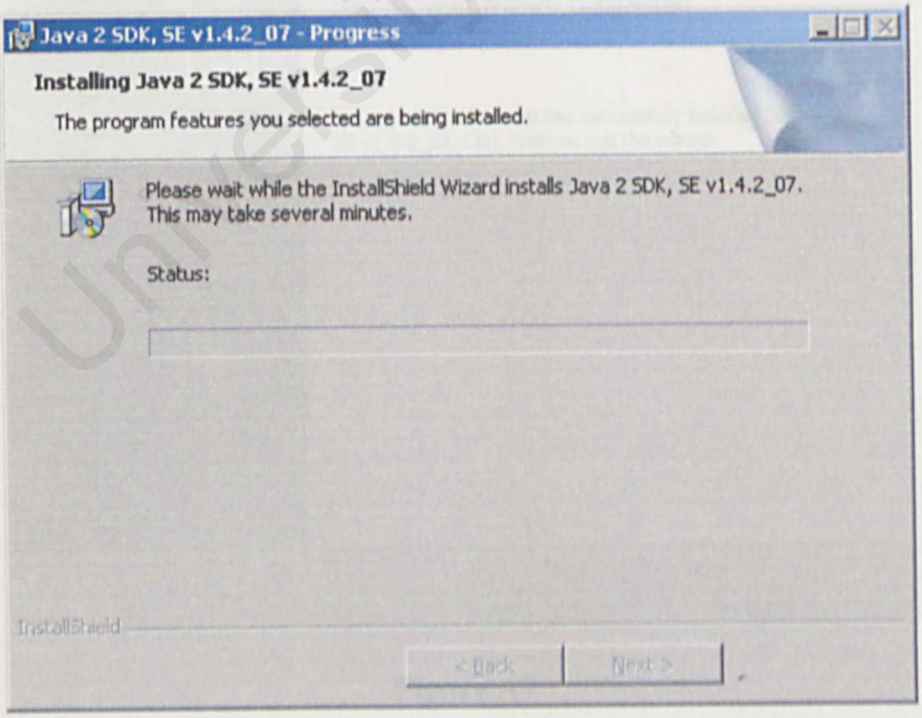

Figure A5

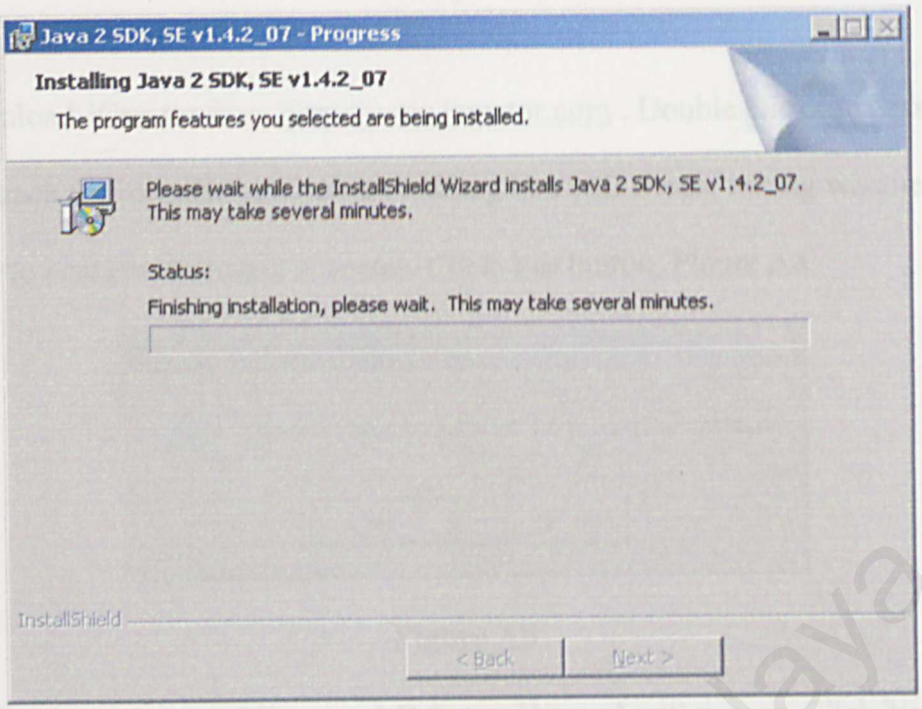

Figure A6

- The **Complete** wizard screen will appear. Click Finish button to finish the installation, **Figure A7.**

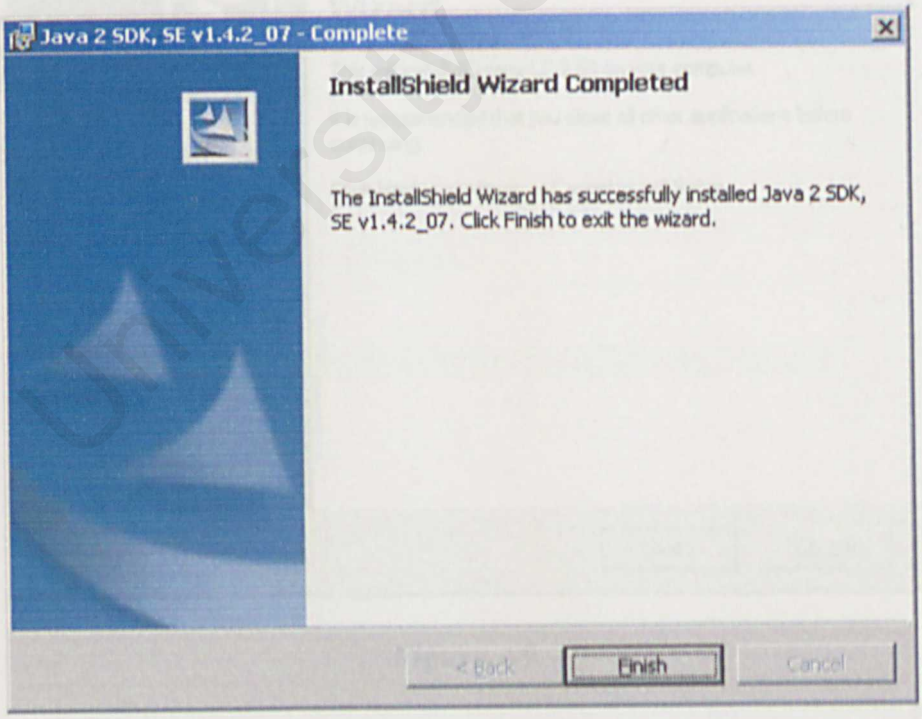

Figure A7

II) Installing JCreator LE version2.5

- Download JCreator from http://www.jcreator.com . Double click the **Setup** icon to launch the installation wizard. A dialog box will appear asking whether you want to continue to install JCreator. Click **Yes** button, Figure A8.

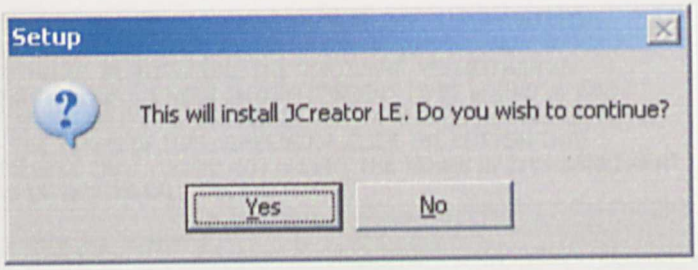

Figure A8

- The **Welcome to the JCreator LE Setup Wizard** will appear. Click **Next** button. See Figure A9.

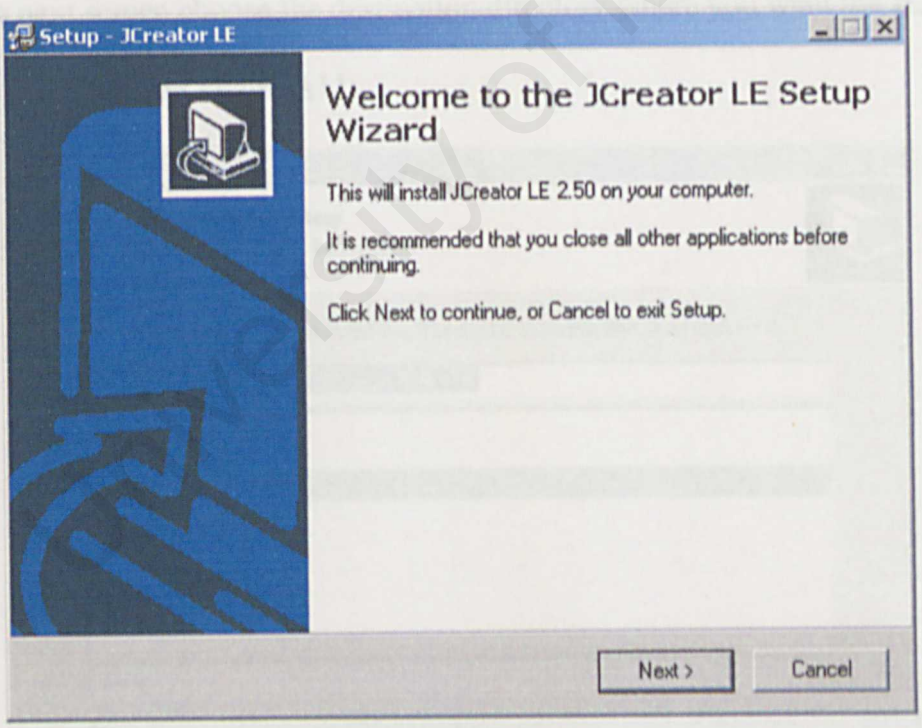

Figure A9

- Click the **I accept the agreement** in the **License Agreement** screen and click the **Next** button. See Figure A10.

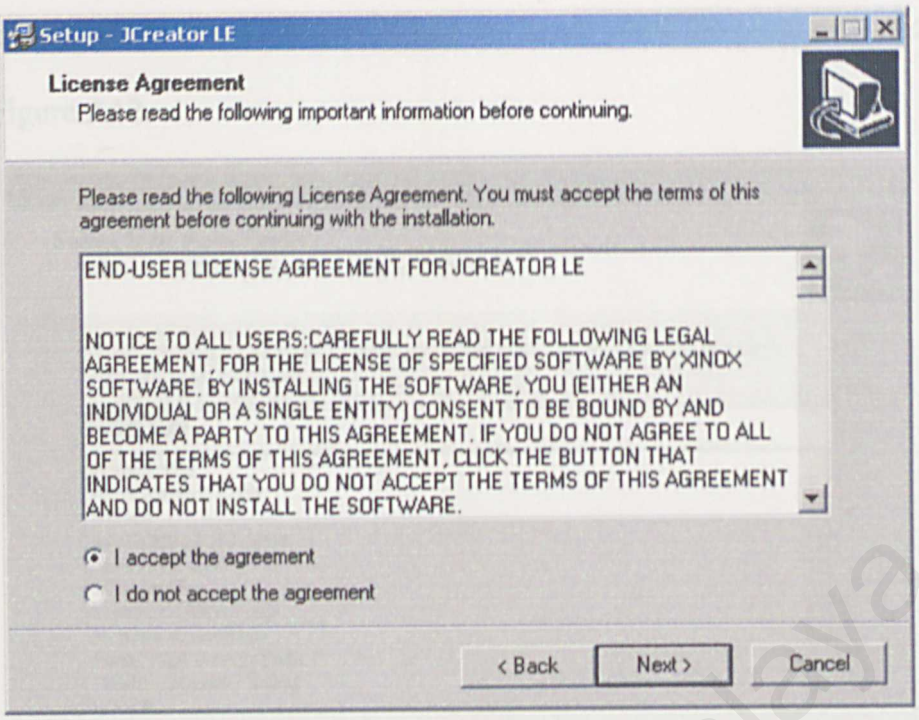

Figure A10

- In the next screen choose the destination directory where you want the JCreator to be installed. See Figure A11.

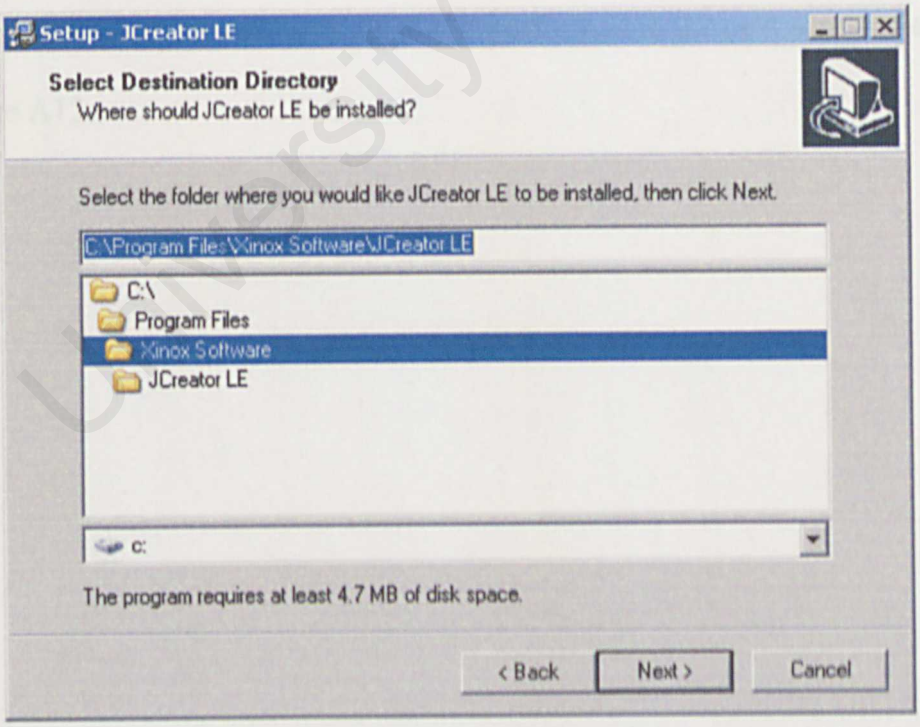

Figure A11

- In the next screen choose where should the Setup place the program's shortcut. See Figure A12.

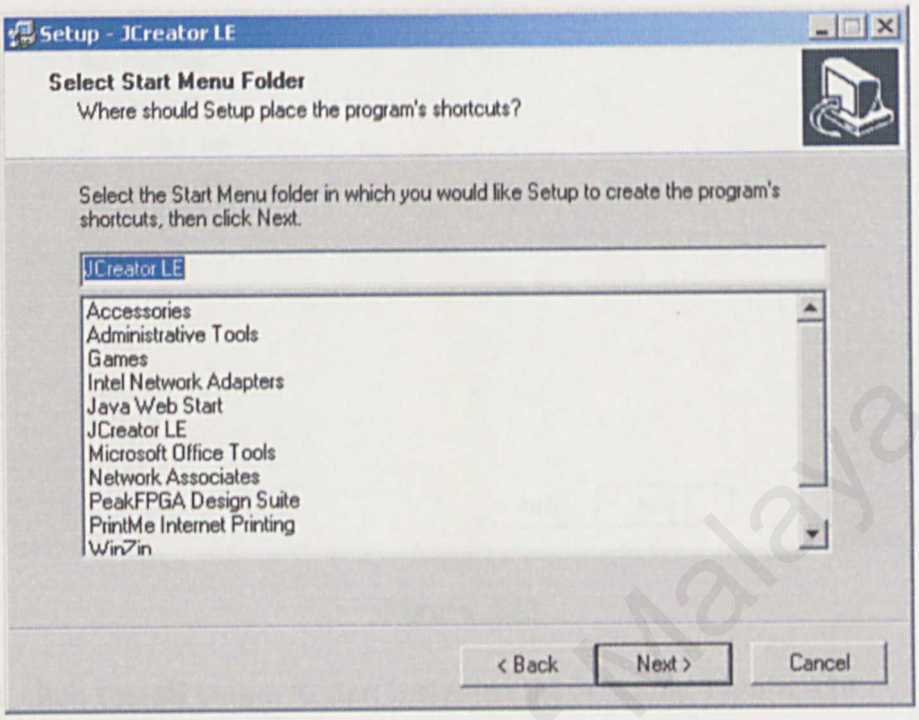

Figure A12

- Choose which additional tasks should be performed in the next screen. See Figure A13.

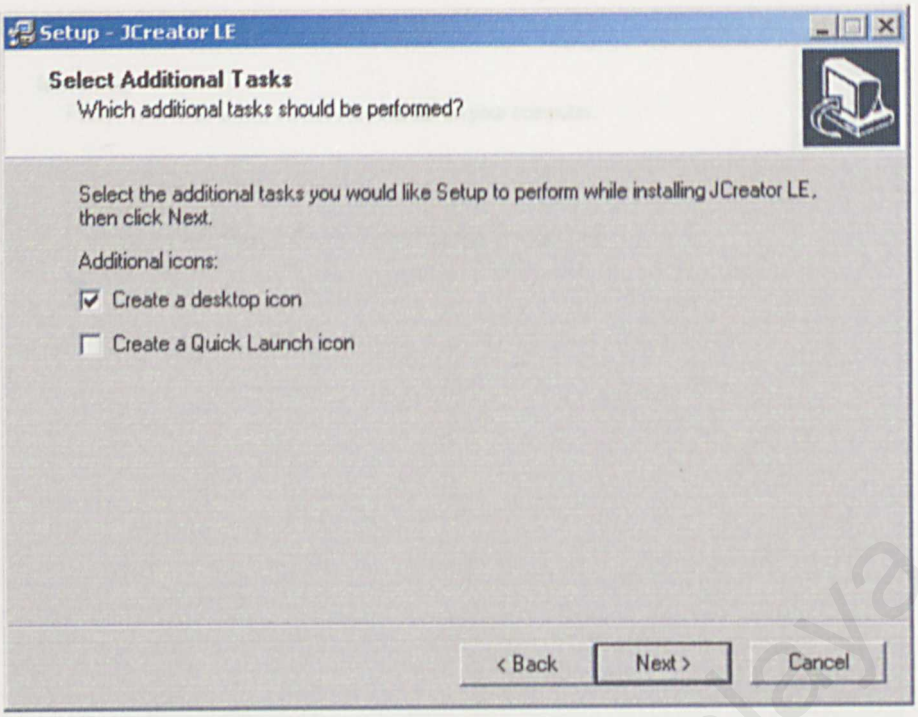

Figure A13

- Next, click **Install** button to start installing the JCreator, Figure A14.

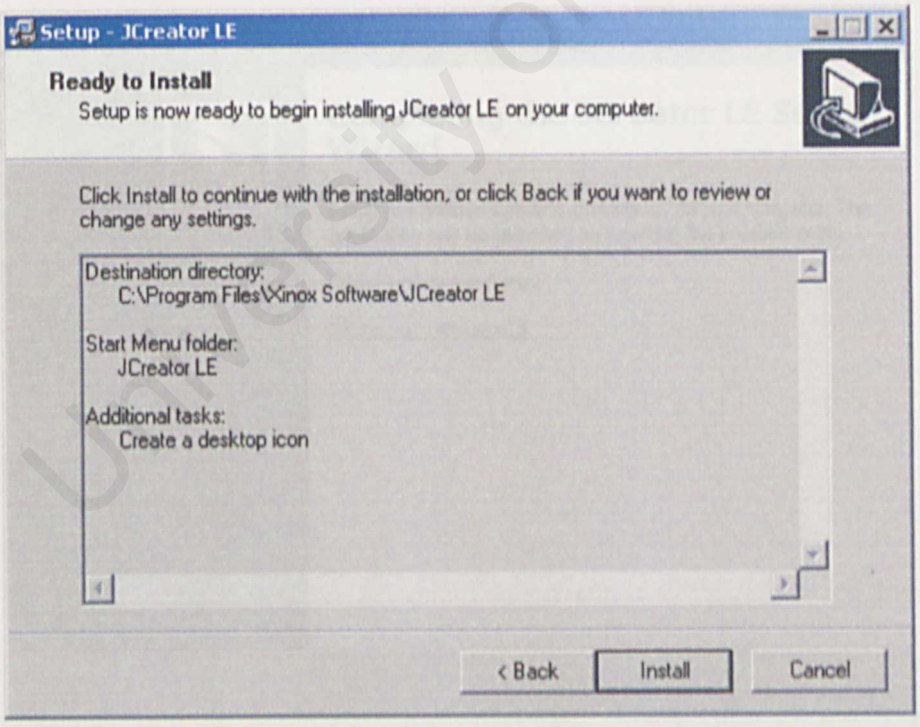

Figure A14

- Please wait while the Setup is installing the JCreator, Figure A15.

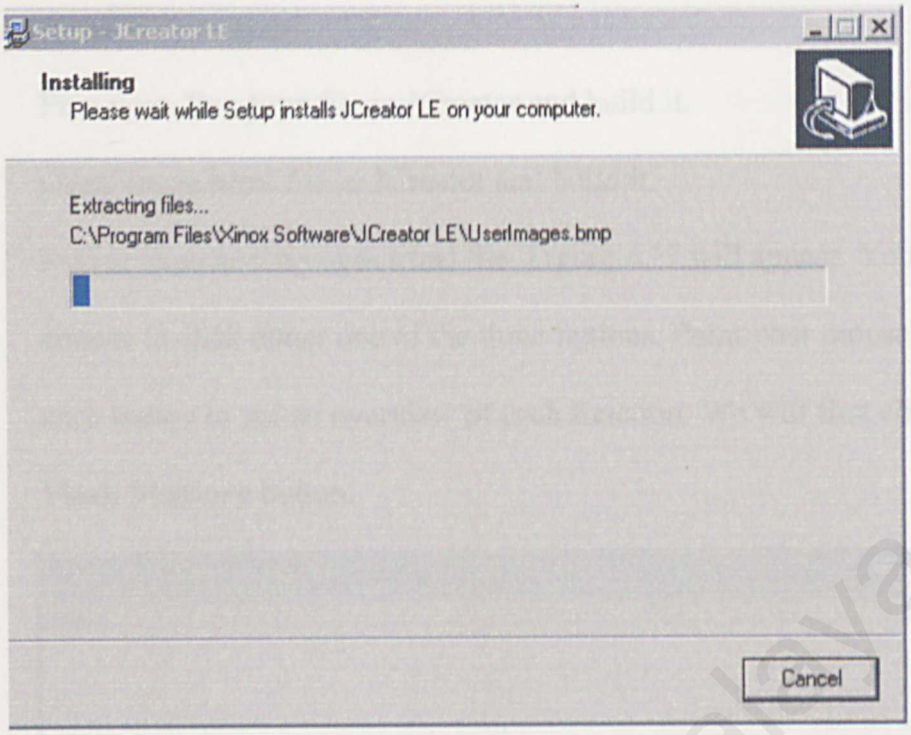

Figure A15

- Lastly click **Finish** button to finish the installation, Figure A16.

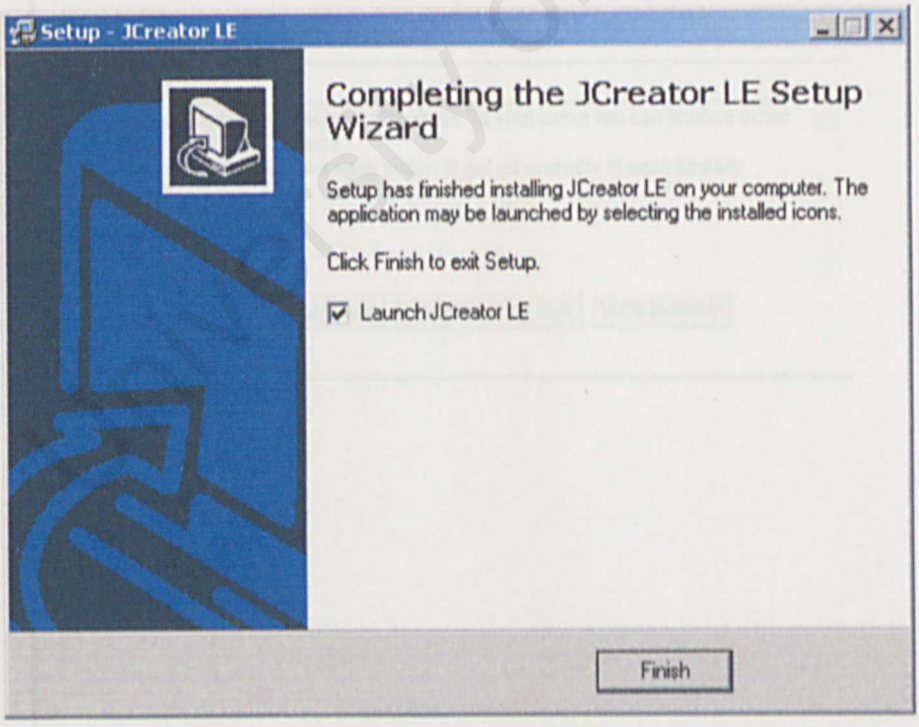

Figure A16

E. Running the program.

1) First open **Box.java** file in JCreator and build it.

2) Open **yoges.html** file in JCreator and build it.

3) Run or execute the **yoges.html** file. Figure A17 will appear. You can choose to click either one of the three buttons. Point your mouse over each button to get an overview of each function. We will first click the **Flash Memory** button.

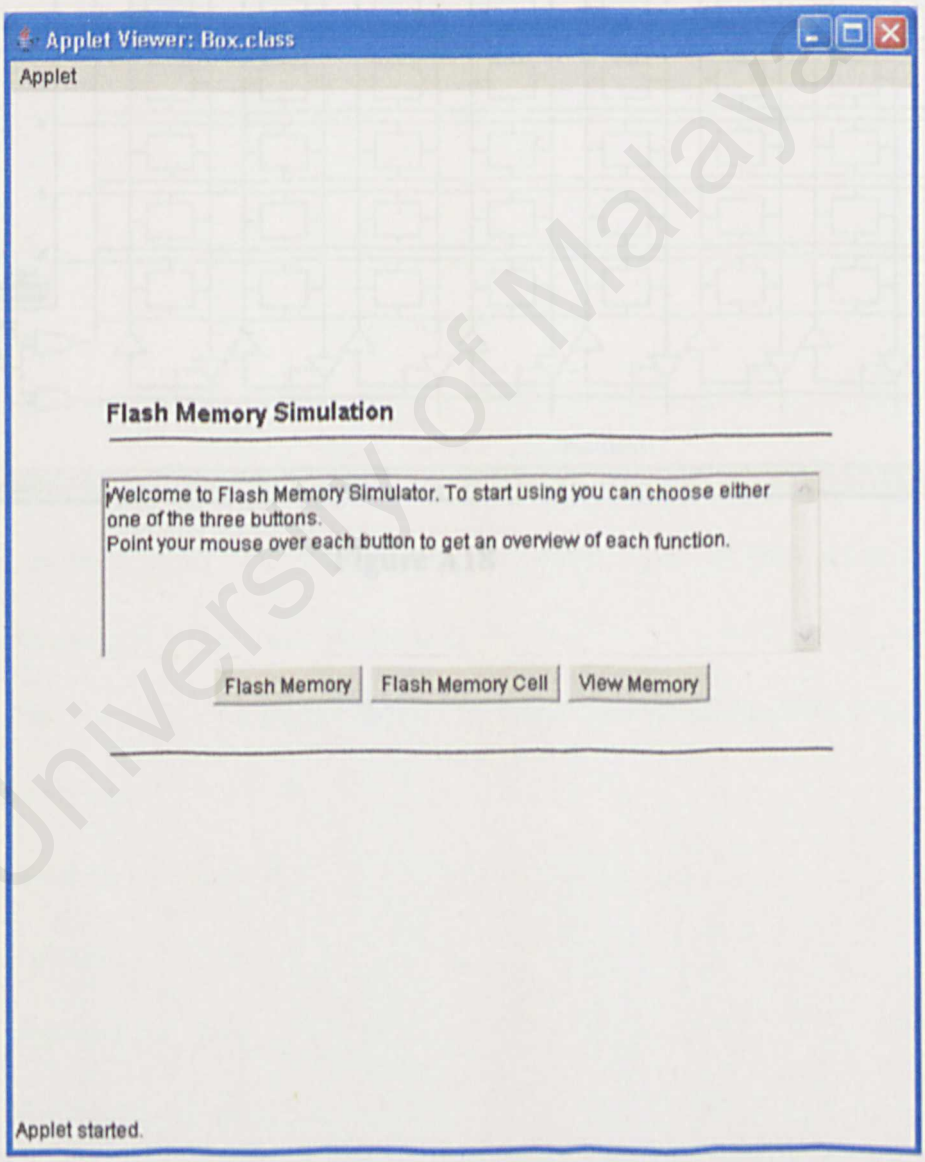

Figure A17

4. Next Figure A18 will appear. Enter the address in the **Address** text field to read the data in the preferred address (0-7) and click the **Read** button, Figure A19.

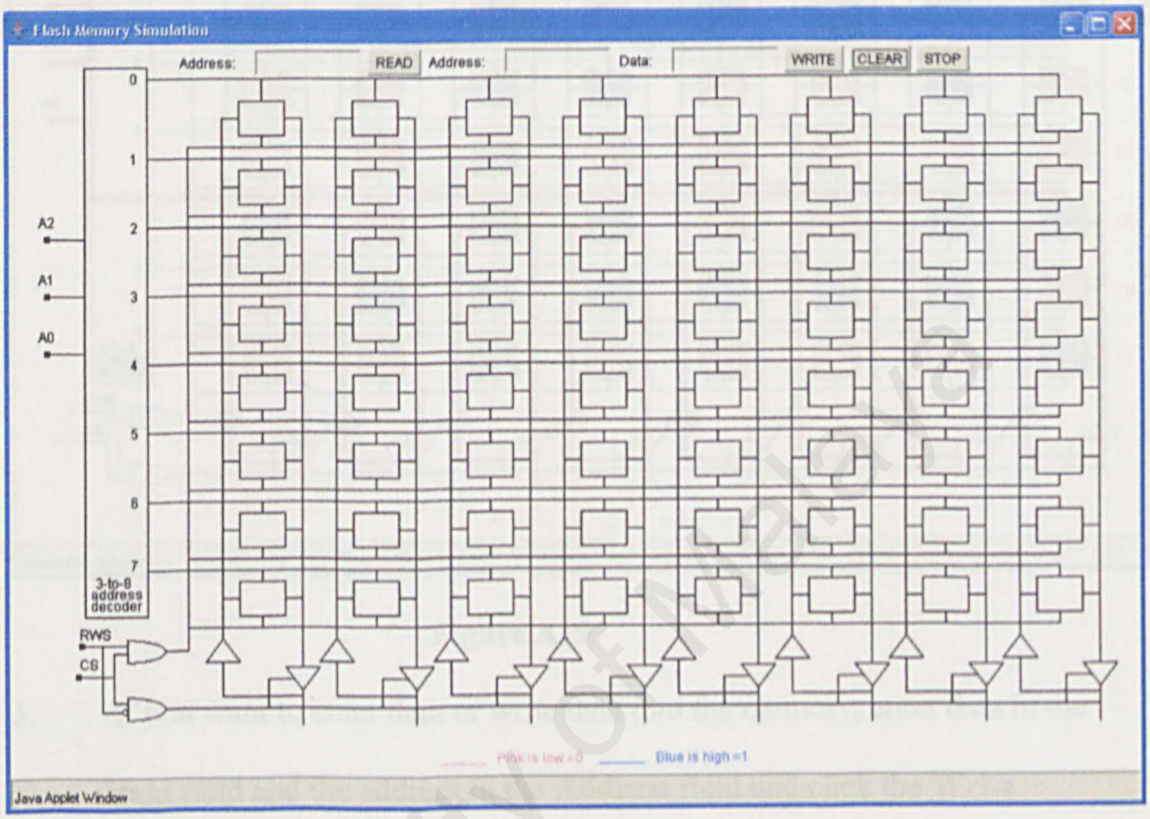

Figure A18

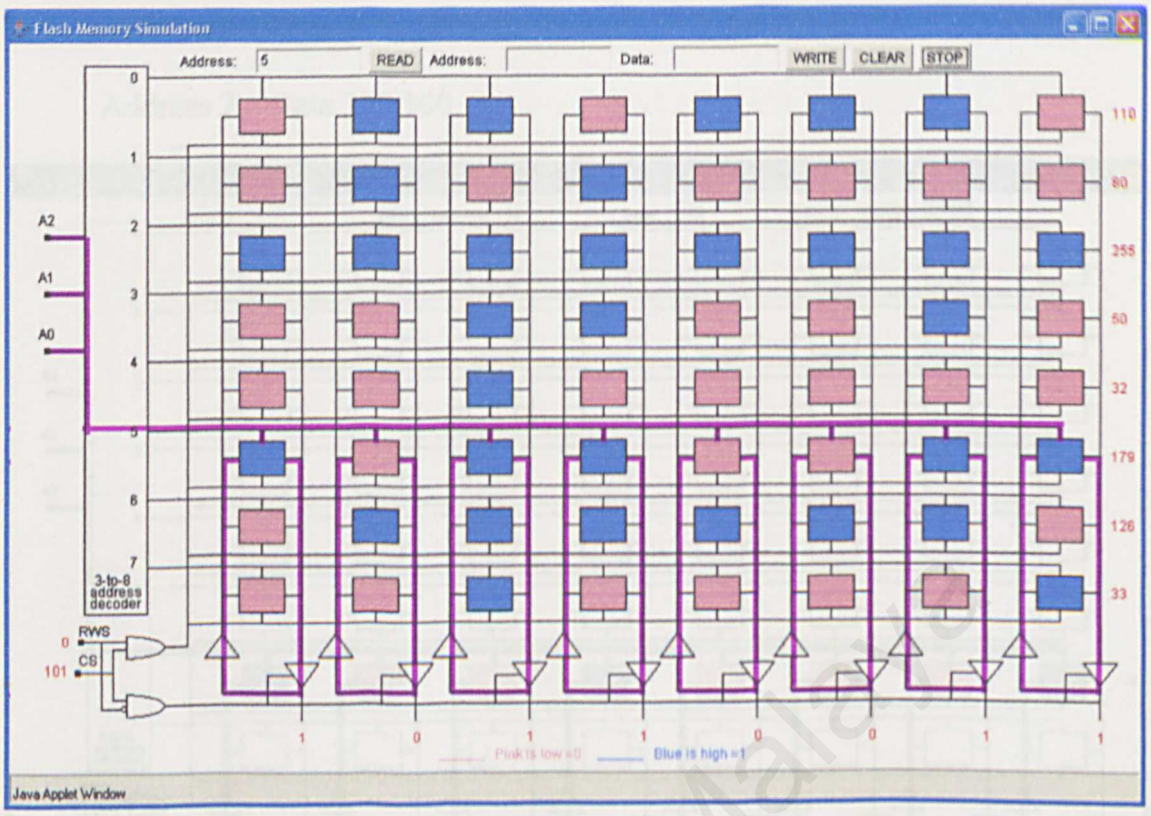

Figure A19

5. If you want to enter data or write data into the memory, enter data in the **Data** field and the address in the **Address** field and click the **Write** Button, Figure A20. You can see the schematic doing the animation. Pink shows zero while blue shows a 1.

The data and address for Write function (already hardcoded inside the program).

Address 0 – Data 0-5

Address 1 – Data 1, 3

Address 2 – Data 2, 4, 5

Address 3 – Data 1, 2

Address 4 – Data 2, 4

Address 5 – Data 210, 90

Address 6 – Data 170, 145

Address 7 – Data 200,100

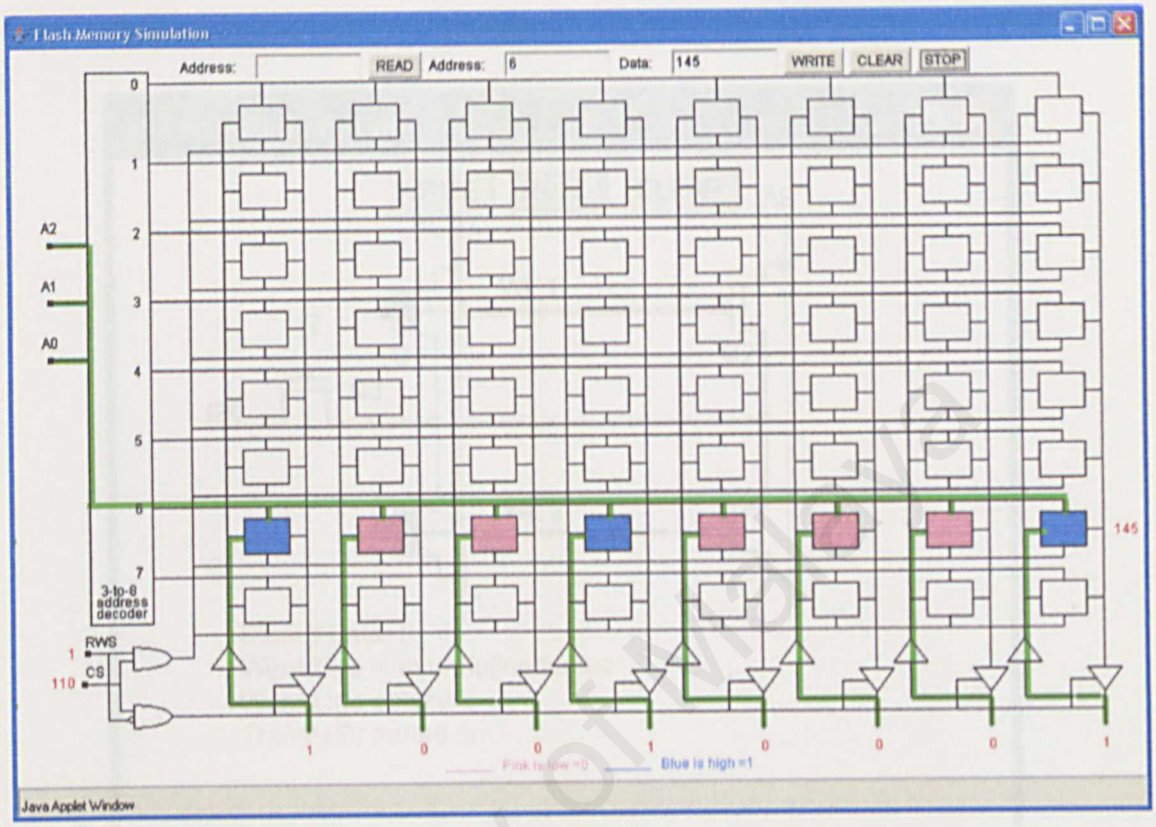

Figure A20

6. If u do not enter any data in the address or the data field and you click Read or Write button, you will get an error as shown in Figure A21.

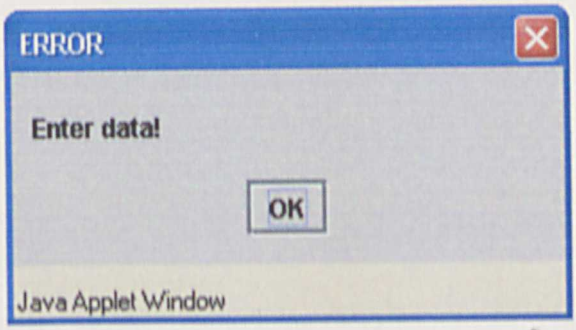

Figure A21

7. When you click the Flash Memory Cell in Figure A22, Figure A23 will appear. You can click the **Read** or **Write** button to see how a cell of Flash works.

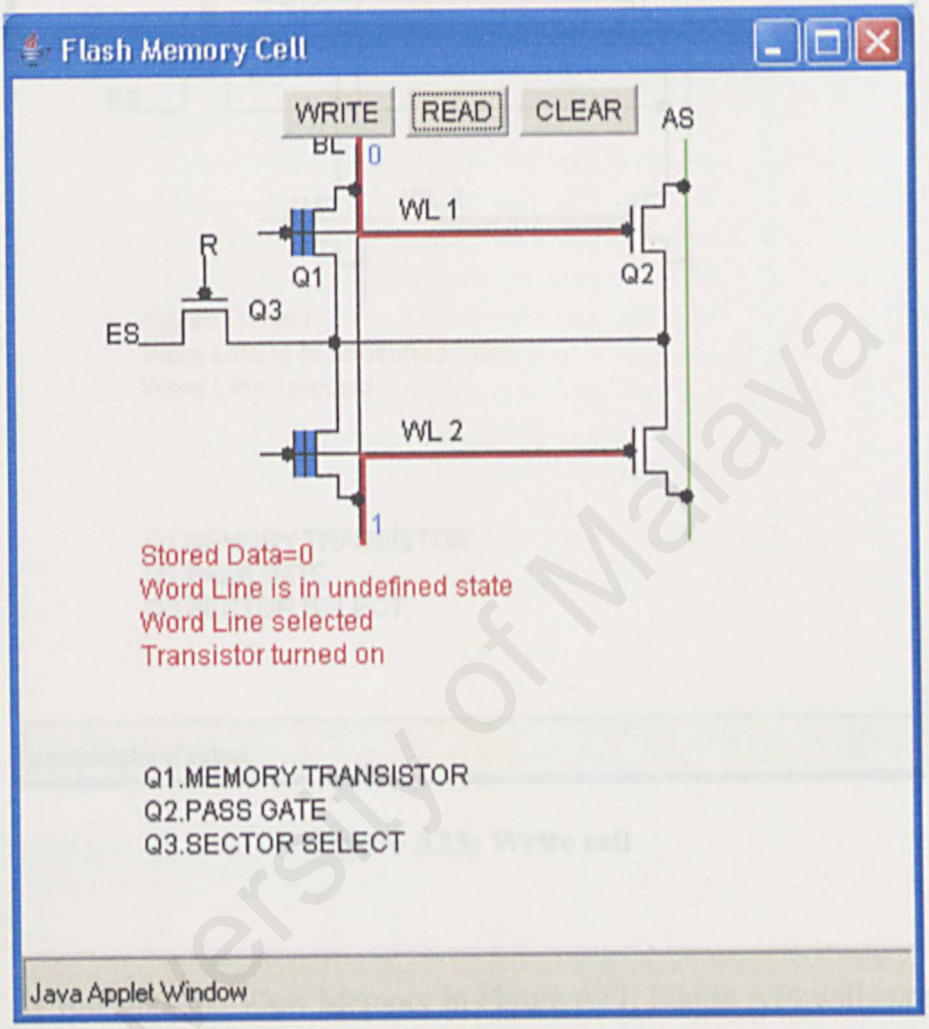

Figure A22: Read cell

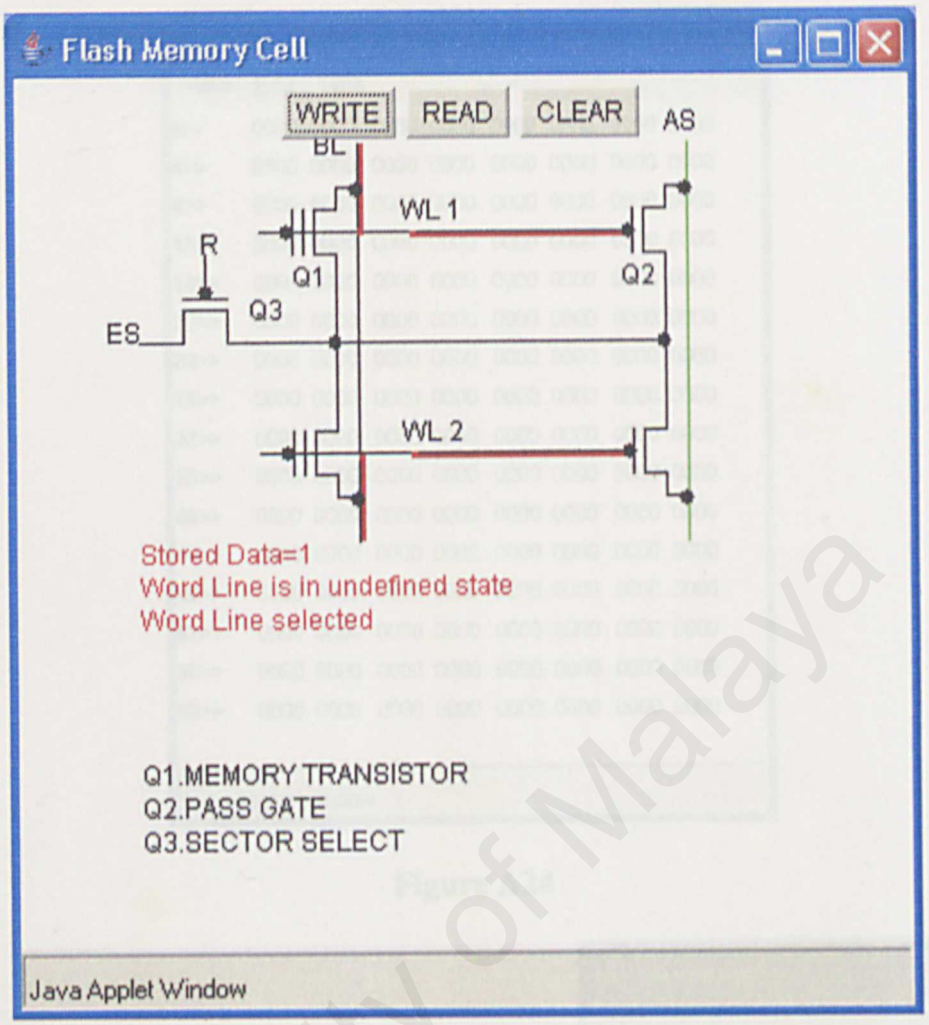

Figure A23: Write cell

8. If you click the View Memory in Figure A17, Figure A24 will appear. It shows the structure of the memory. You can enter and edit data through the Edit menu. Choose either Edit Memory, Fill Memory or Clear Memory. Figure A25-Figure A26. Data can be viewed in binary or hexadecimal.

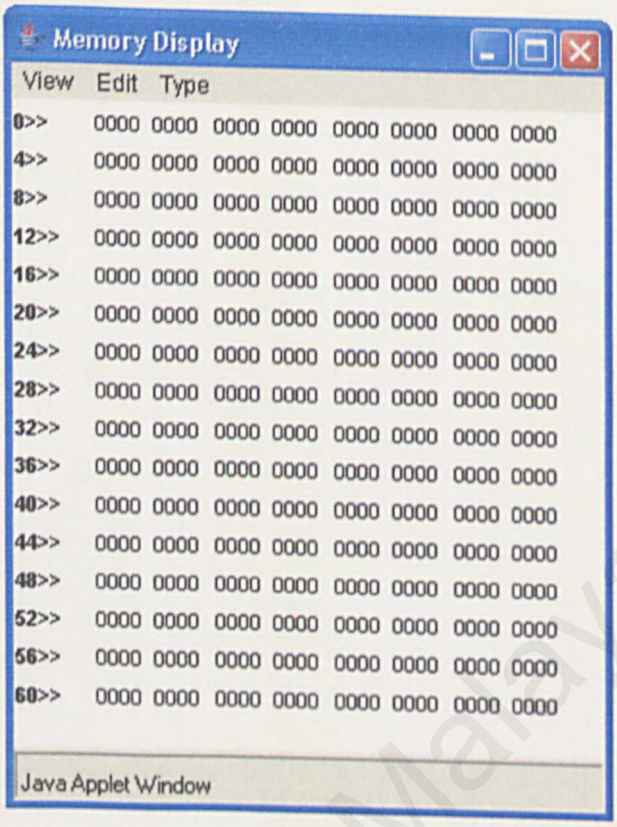

Figure A24

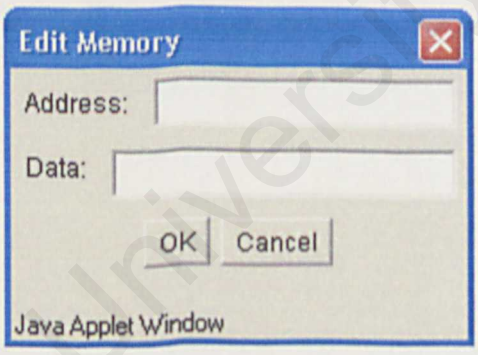

Figure A25

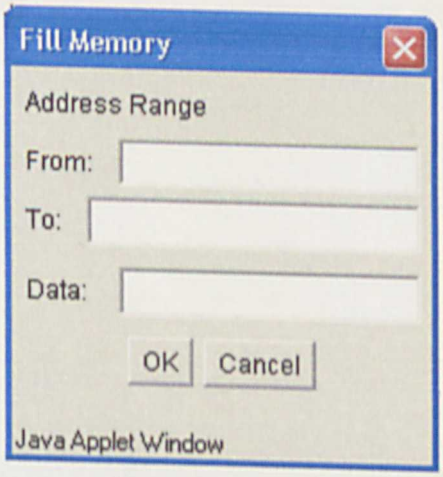

Figure A26